TAMPEREEN TEKNILLINEN YLIOPISTO

NIKO HÄIKIÖ

ECLIPSE-ALUSTAAN PERUSTUVA INTEGROITU KEHITYSYMPÄRISTÖ SQF-OHJELMOINTIKIELELLE

Diplomityö

Tarkastaja: Professori Hannu Koivisto
Tarkastaja ja aihe hyväksytty
Teknisten tieteiden tiedekuntaneuvoston kokouksessa
6. huhtikuuta 2016

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
HÄIKIÖ, NIKO: Eclipse-alustaan perustuva integroitu kehitysympäristö SQF-ohjelmointikielelle
Diplomityö, 77 sivua, 0 liitesivua
Marraskuu 2016
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Hannu Koivisto
Avainsanat: ANTLR, Eclipse, IDE, Integroitu kehitysympäristö, SQF, Xtext

Integroidut kehitysympäristöt ovat olleet oleellisessa osassa ohjelmistokehitystä jo vuosikymmenien ajan. Tämän työn tavoitteena on tutkia niiden käyttöä, historiaa ja rakennetta. Saadun tiedon perusteella rakennetaan uusi integroitu kehitysympäristö SQF-ohjelmointikielelle.

Integroituihin kehitysympäristöihin tutustutaan yleisellä ja yksityiskohtaisella tasolla. Ensimmäisen vaiheessa määritellään integroidun kehitysympäristön yleiset ominaisuudet ja seuraavassa vaiheessa analysoidaan olemassa olevia integroituja kehitysympäristöjä. Lisäksi esitellään integroitujen kehitysympäristöjen käytöstä tehtyä aikaisempaa tutkimusta.

Työn tuloksena on uusi integroitu kehitysympäristö nimeltään Side. Side on SQF-ohjelmointikielelle suunniteltu integroitu kehitysympäristö Eclipse-alustan päälle. Eclipse-alustan lisäksi hyödynnettiin Xtext-kehystä kielentunnistajan kehityksessä. SQF on Arma 3 -pelin laajentamiseen tarkoitettu toimialakohtainen ohjelmointikieli, minkä vuoksi tässä työssä tutustutaan myös Arma 3 -peliin, pelien modaamiseen ja SQF-ohjelmointikieleen.

Tutkimuksessa havaittiin, että integroitujen kehitysympäristöjen ominaisuudet ovat yleisessä käytössä ohjelmistokehittäjien keskuudessa. Xtext ja Eclipse soveltuvat hyvin integroidun kehitysympäristön kehitykseen, jos ohjelmointikieli täyttää tietyt reunaehdot ja lievät tehokkuusongelmat ovat siedettävissä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
HÄIKIÖ, NIKO: An Eclipse-Based Integrated Development Environment for SQF
Master of Science Thesis, 77 pages, 0 Appendix pages
November 2016
Major: Automation software engineering
Examiner: Professor Hannu Koivisto
Keywords: ANTLR, Eclipse, IDE, Integrated Development Environment, SQF, Xtext

Integrated development environments (IDE's) have been popular part of software development for several decades. This thesis focuses on the history, usage and structure of an IDE. On the basis of the information acquired a new IDE called Side is developed.

Integrated development environments are discussed both in general terms and in detail. In the general discussion, a consensus of common features of an IDE is constructed. In the detailed discussion, a set of already existing IDE's is analyzed while trying to figure out the source of their popularity. In addition, a study regarding the usage of an IDE is introduced.

A new IDE called Side was developed during this thesis. Side is an IDE for SQF programming language. It is designed to work on Eclipse-platform while Xtext framework was used in developing the language recognizer. SQF is a programming language for the modification of Arma 3, hence the general traits of SQF, Arma 3 and game modification are also discussed.

It was concluded that IDE based features are commonly utilized by programmers. Xtext and Eclipse are suitable for IDE development if the target language meets certain pre-conditions and minor performance costs are considered sustainable.

ALKUSANAT

Diplomityö on kehitetty omakohtaisen idean pohjalta, jonka motivaationa on toiminut kokemus SQF-ohjelmoinnin haasteita ja halusta lieventää niitä. Vallitsevana ajatuksena diplomityön teon aikana on ollut halukkuus ratkoa ongelmia huolimatta niiden monimutkaisuudesta tai haasteellisuudesta.

Haluan kiittää työni ohjaajaa, tekniikan tohtori Timo Vepsäläistä suuntaviivojen antamisesta, ammattitaitoisesta ja kärsivällisestä ohjaamisesta sekä kannustavasta asenteesta. Kiitos kuuluu myös työn edelliselle tarkastajalle, emeritus professori Seppo Kuikalle ja nykyiselle tarkastajalle professori Hannu Koivistolle epätavanomaisen aiheen tuomitsemattomasta hyväksymistä sekä myös kannustavasta asenteesta. Kiitos myös kaikille muille työn edistymistä edesauttaneille henkilöille.

Tampereella 23.11.2016,

_____

Niko Häikiö

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

ANTLR	Kielentunnistimen generoija (engl. Another Tool For Language Recognition)
Avustaja	Ohjelmiston osa, joka tarjoaa ohjatun luomisen jollekin ohjelmiston oliolle.
C++	Eräs ohjelmointikieli, joka kääntyy konekieleksi.
DLL	Windowsin dynaaminen kirjasto (engl. dynamic-link library)
Eclipse	Ohjelmistoalusta erityisesti integroitujen kehitysympäristöjen toteuttamiseen
EMF	Datamallin rakentamiseen käytettävä kehys. (engl. Eclipse Modeling Framework)
gSoap	Työkalu Web service -sovelluksien kehittämiseen C++- ja C -kielillä.
IDE	Integroitu kehitysympäristö (engl. integrated development environment)
Java	Eräs ohjelmointikieli, joka kääntyy tavukoodiksi.
Kielentunnistaja	Ohjelmiston osa, joka rakentaa kirjoitetusta lähdekoodista tietorakenteen ajonaikaiseen muistiin.
Komentosarja	Pätkä lähdekoodia, jonka voi suorittaa sellaisenaan
Kääntäminen	Ohjelmointikielen muuttaminen toiseen muotoon esim. konekieleksi
Literaali	Tiettyä tietotyyppiä edustavan tiedon esitysmuoto
Modaus	Modien kehittäminen
Modi	Yhteisön kehittämä laajennus peliohjelmistoon
PBO	Arma 3 -pelin käyttämä tiedostojen arkistointimuoto (engl. packed bank of files)
Refaktorointi	Lähdekoodin muokkaus niin, että vain laadulliset ominaisuudet muuttuvat.
RPT-loki	Arma 3 -pelin generoima loki.
SQF	Ohjelmointikieli, jota käytetään Bohemia Interactive Studion pelien toiminnallisuuden laajentamiseen yhteisön toimesta.
Syntaksin tarkastaja	Ohjelmiston osa, joka tarkastaa kirjoitetun lähdekoodin syntaksin.
Säie	Ohjelmakoodia suorittava entiteetti

Tooltip	Teksti-ikkuna, joka kertoo lisätietoja käyttöliittymäoliosta sen lähellä.
Tunnus	Lähdekoodin osan nimi esimerkiksi muuttujan, funktion tai luokan nimi
Valevirhe	Syntaksin tarkastajan virheellisesti ilmoittama virhe
Varattu sana	Sana, jota ei voida käyttää tunnuksena.
Visual Studio	Microsoftin julkaisema integroitu kehitysympäristö
XML	Merkkikieli rakenteellisen datan tallentamiseen (engl. Extensible Markup Language)
Xtext	Kehys ohjelmointikielien määrittelemiseksi

1 JOHDANTO

Integroidulla kehitysympäristöllä tarkoitetaan lähdekoodin muokkaukseen tarkoitettua ohjelmistoa, johon on koottu monia ohjelmointiin liittyviä työkaluja yhden kokonaisuuden alle. Jo 80-luvulla huomattiin, että erinäisten työkalujen kuten kääntäjän, editorin ja lähdekoodianalysaattoreiden käyttö helpottui ja yleistyi, kun niiden kaikkien käyttö pystyttiin tekemään yhden ohjelman toimesta.

Tässä diplomityössä tarkastellaan integroitujen kehitysympäristöjen merkitystä ohjelmistokehityksessä analysoiden olemassa olevia kehitysympäristöjä ja niiden käyttöä. Tutkimustuloksien perusteella muodostetaan vaatimukset kehittävälle SQF-ohjelmointikielen integroidulle kehitysympäristölle (engl. integrated development environment, lyh. IDE). SQF on Bohemia Interactive Studion kehittämien pelien laajentamiseen tarkoitettu ohjelmointikieli. SQF-lähdekoodi käännetään ja ajetaan pelin sisällä, mikä asettaa erilaisia vaatimuksia integroidun kehitysympäristön toteutukselle. Tilannetta voidaan verrata sulautettuihin järjestelmiin, joissa ohjelmakoodin todellinen toiminta nähdään vasta, kun se ajetaan itse laitteella. Lähdekoodin siirtäminen ja ajaminen pelin sisällä on hankalaa sekä hidasta, josta johtuen ohjelmointivirheiden olemassaolo hidastaa huomattavasti ohjelmistokehitystyötä.

1.1 Tutkimuskysymykset ja -tavoitteet

Diplomityön tarkoituksena on tutkia integroitujen kehitysympäristöjen käyttöä ja kehitystä. Tutkimuskysymyksinä ovat:

1. Miten ohjelmistokehittäjät hyödyntävät integroituja kehitysympäristöjä?
2. Mitä käyttäjävaatimuksia tulisi olla SQF- ja Arma 3 -kohtaisella integroidulla kehitysympäristöllä?
3. Miten Eclipse-alustan päälle voidaan kehittää integroitu kehitysympäristö?

Ensimmäiseen tutkimuskysymykseen pyritään vastaamaan kirjallisuusselvityksen avulla. Toiseen tutkimuskysymykseen vastataan sekä toteutusosan että kirjallisuusselvityksen avulla. Kirjallisuuden avulla on tarkoitus tutkia alustavasti, mitä vaatimuksia integroidulle kehitysympäristölle tulisi asettaa, ja toteutusosalla varmennetaan näiden vaatimuksien riittävyys. Kolmanteen tutkimuskysymykseen vastataan pelkästään toteutusosan avulla. Diplomityön toteutusosana tehtävä integroitu kehitysympäristö pyrkii ly-

hentämään merkittävästi SQF-kehitystyöhön kuluvaa aikaa seuraavien toimenpiteiden avulla:

- Lähdekoodin siirtoajan lyhentäminen pelin ja editorin välillä
- Syntaktisten virheiden ilmoitus
- Huonojen ohjelmointitapojen ilmoitus
- Lähdekoodin generointi ja automaattinen täydennys

Nämä ominaisuudet pyritään kehittämään mahdollisimman helppokäyttöiseen integroituun kehitysympäristöön nimeltään Side, joka on lyhenne sanoista ”SQF Integrated Development Environment”. Syntaksin tarkistuksella pyritään estämään syntaktisten ohjelmointivirheiden syntyminen. Syntaksin tarkistuksen lisäksi lähdekoodin tulkinnalla pyritään ilmoittamaan niin sanotuista huonoista ohjelmointikäytännöistä. Automaattisella täytöllä mahdollistetaan intuitiivinen lähdekoodin muokkaus ja näin nopeutetaan kehitystyötä. Usein toistuville SQF-ohjelmoinnin rutiineille pyritään kehittämään yksinkertainen lähdekoodigeneraattori. Integraatiolla pelin kanssa pyritään helpottamaan ohjelmakoodin siirtämistä editorista peliin ja sen ajamista.

1.2 Lähestymistapa

Tutkimustyön konkretisoimiseksi kehitetään SQF-ohjelmointikielelle integroitu kehitysympäristö nimeltään Side. Aluksi todetaan integroidun kehitysympäristön tarve, jonka jälkeen huomioidaan, että olemassa olevat editorit eivät vastaa kaikkien SQF-ohjelmoijien tarpeita. Side tullaan kehittämään Eclipse-alustan päälle.

Side-kehitysympäristön kehittämisessä sovelletaan ketteriä kehitysmenetelmiä, lean-periaatetta, olio-ohjelmointia ja testivetoista kehitystä. Kokonaisuudessaan kehitysprosessi noudattaa suurelta osin Extreme Programming -metodologiaa (lyh. XP). Mahdollisimman hyvän vaatimusmäärittelyn saavuttamiseksi tehdään analyysiä olemassa olevien integroitujen kehitysympäristöjen välillä ja tutkitaan olemassa olevia SQF-editoreita. Lisäksi tutkitaan integroitujen kehitysympäristöjen historiaa ja käyttöä kirjallisuuden kautta. Edellä mainituilla menetelmillä pyritään vastaamaan toiseen ja kolmanteen tutkimuskysymykseen.

Diplomityön kirjallinen osuus koostuu kahdestatoista luvusta. Ensimmäinen luku on johdanto. Toisessa luvussa käsitellään integroituja kehitysympäristöjä yleisellä tasolla tutkimalla niiden historiaa ja käyttöä. Luvun tarkoitus on muodostaa käsitys integroitujen kehitysympäristöjen merkityksestä ohjelmistokehityksessä ja täten vastata ensimmäiseen tutkimuskysymykseen. Kolmannessa luvussa tutkitaan yleisiä integroituja kehitysympäristöjä ja olemassa olevia SQF-editoreita, minkä perusteella pyritään havaitsevaan hyvän integroidun kehitysympäristön ominaisuudet. Neljännessä luvussa kerrotaan pelien yhteisöpohjaisesta muokkauksesta eli modaamisesta ja sen suhteesta Arma 3 -pe-

liin. Viidennessä luvussa kerrotaan SQF-ohjelmointikielen perusteista ja miten se toimii yhdessä Arma 3 -pelin kanssa. Luvut 2-5 muodostavat yhdessä teoreettisen pohjan Side-kehitysympäristön vaatimuksille. Kuudennessa ja seitsemännessä luvussa tutkitaan Eclipse-alustaa, sen päällä toimivaa Xtext-kehystä ja muodostetaan konsensus siitä, miten Eclipse-alustan päälle saadaan rakennettua integroitu kehitysympäristö. Kahdeksannessa luvussa esitellään käytetyt ohjelmistotuotannon menetelmät. Yhdeksännessä luvussa tehdään Side-kehitysympäristön määrittely ja vaatimuslistaus. Luvussa hyödynnetään edellisten lukujen tutkimustuloksia vaatimuslistauksen muodostamisessa ja täten vastataan toiseen tutkimuskysymykseen. Kymmenennessä luvussa suunnitellaan ohjelmisto. Kuudennen ja seitsemännen luvun tietoja käytetään suunnittelun tukena. Yhdennessätoista luvussa kerrotaan, miten Side on toteutettu. Kahdennessatoista luvussa pohditaan, täyttikö Side sille asetetut vaatimukset, esitetään lisäkehitysideoita ja tehdään diplomityön johtopäätökset. Myös Side-kehitysympäristön tulevaa elinkaarta pohditaan.

2 INTEGROITU KEHITYSYMPÄRISTÖ

Viimeisten vuosikymmenien aikana monenlaisia työkaluja on kehitetty ohjelmistokehityksen tehostamiseksi. Työkalujen käyttöä on haitannut niiden rajoittunut sovellusalue, esimerkiksi pelkästään dokumentaation toteutus. Tämän ongelman ratkaisemiseksi on kehitetty ohjelmistoja, jotka integroivat eri sovellusalueiden työkalut yhden kokonaisuuden alle. Tätä kokonaisuutta kutsutaan integroiduksi kehitysympäristöksi. [2]

2.1 Integroitujen kehitysympäristöjen tavoitteet

Integroidut kehitysympäristöt kuten Visual Studio kykenevät automatisoimaan ominaisuuksiensa kautta ohjelmointiin liittyviä tehtäviä. Näihin ominaisuuksiin kuuluvat lähdekoodin refaktorointi, automaattinen täydennys ja käännösvirheiden korjaus ehdotuksien avulla [5]. Ominaisuuksilla on kaksi tavoitetta: ohjelmistokehityksen nopeuttaminen ja ohjelmointivirheiden minimointi. Ominaisuuksien käyttö on yleistä normaalien tekstinmuokkaustoimintojen kuten tallennuksen, liittämisen ja kopioinnin ohella [1].

2.2 Integroitujen kehitysympäristöjen ominaisuudet

Integroidut kehitysympäristöt sisältävät monia ominaisuuksia, joilla pyritään vähentämään ohjelmoijalle asetettua taakkaa. Seuraavat neljä ominaisuutta ovat yleisiä integroiduissa kehitysympäristöissä.

1. **Lähdekoodin tulkinta**: Lähdekoodin tulkinta tarkoittaa integroidun kehitysympäristön kykyä tunnistaa lähdekoodin kannalta tärkeät termit kuten funktioiden ja muuttujien tunnukset. Tämä mahdollistaa termien värittämisen lähdekoodin luettavuuden takaamiseksi, virhetarkastuksen ja mahdollisten termien ehdottamisen ohjelmoijalle eli automaattisen täydennyksen.
2. **Resurssienhallinta**: Resurssienhallinnalla tarkoitetaan integroidun kehitysympäristön kykyä hallita tarjolla olevia resursseja kuten ohjelmistokirjastoja, lähdekooditiedostoja, binääritiedostoja ja niihin liittyviä kansioita. Integroitu kehitysympäristö vähentää näin useasta tiedostosta johtuvan työkuorman määrää ohjelmistokehittäjälle.
3. **Virheidenetsintätyökalut**: Virheidenetsintätyökalut (engl. debug tools) mahdollistavat ohjelman virheiden etsinnän. Ne voivat esimerkiksi mahdollistaa muuttujien arvojen ajonaikaisen tarkastelun ja muuttamisen. Ohjelman suorituksen

hallinnalla voidaan ohjelmaa suorittaa käsky kerrallaan ja suoritukseen voidaan asettaa pysähdyskohtia (engl. breakpoints).

4. **Lähdekoodin käännöstyökalut**: Käännöstyökalujen avulla lähdekoodi muunnetaan ajettavaksi ohjelmaksi. Käännöstyökalut mahdollistavat lähdekoodin kääntämisen toiseen kieleen. Yleensä tämä tarkoittaa ihmiselle selkeälukuisen ohjelmointikielen kääntämistä konekieleksi. [5]

Tätä listaa pyritään laajentamaan tutkimalla olemassa olevia integroituja kehitysympäristöjä luvussa 3.

2.3 Integroidun kehitysympäristön historia

Integroitujen kehitysympäristöjen tarve tiedostettiin 80-luvulla. Tuolloin käyttöjärjestelmät eivät tukeneet moniajoa, minkä takia ohjelman suorittamiseksi ohjelmoijat joutuivat ensin sulkemaan editorin ja ajamaan kääntäjän. Virheen ilmetessä jouduttiin ohjelman tiedot kopioimaan paperille, minkä jälkeen virhe voitiin etsiä editorilla lähdekoodista. [4] Työkalujen suuri määrä aiheutti myös ongelmia ohjelmistokehityksessä. Ratkaisuksi esitettiin integroitua kehitysympäristöä, joka kokoaisi työkalut yhden ohjelman alle. [3]

Vuonna 1983 Borland Ltd. sai käsiinsä Pascal-kääntäjän tanskalaiselta ohjelmoijalta nimeltään Anders Hejlsberg ja julkaisi sen yhdysvalloissa nimellä TurboPascal. Ohjelmisto sisälsi editorin ja kääntäjän. Yksi aikaisemmista käyttäjistä oli Microsoft-yrityksessä työskentelevä Charles Patzold, joka kehui editorin kykyä osoittaa virheet ja ohjata ohjelmoijan virheen sijaintiin editorissa. Myös pitkäaikainen Borlandin developer relations -osaston johtaja David Intersimone kehui TurboPascalia sen kääntäjän ja virheentarkastajan ansiosta. [4]

Microsoft julkaisi vuonna 1985 Windowsin, mutta vasta vuonna 1991 Windows 3 -julkaisun jälkeen sen käyttö yleistyi. Tämä johti ohjelmistojen graafisuuden lisääntymiseen, mikä lopulta johti siihen, että myös ohjelmoijat halusivat graafisia työkaluja. Vuonna 1991 julkaistiin Visual Basic (VB), jota pidetään ensimmäisenä aitona integroituna kehitysympäristönä. Ohjelmoija ja Visual Developer lehden julkaisijan Jeff Duntemannin mukaan tämä lisäsi merkittävästi ohjelmointitehokkuutta. [4]

Bortland huomasi Visual Basicin julkaisun nopeasti ja haastoi sen kehittämällä Pascal-ohjelmointikielelle oman integroidun kehitysympäristön, Delphin, vuonna 1992. Delphi mahdollisti ohjelmiston suunnittelun, ohjelmoinnin, testauksen ja virheiden etsinnän yhdellä ohjelmistolla. Lisäksi Delphi-kehitysympäristössä oli ominaisuus, joka mahdollisti ohjelmiston kokoamisen valmiista objekteista. Myöhemmin Delphi-kehitysympäristöön julkaistiin myös C++-tuki. [4]

Vuosien saatossa integroidut kehitysympäristöt ovat helpottaneet ohjelmointia. Ohjelmistokehityksen helppous on johtanut siihen, että vähäisellä koulutustaustalla voidaan toteuttaa ohjelmia, jotka toimivat, vaikka toteutus olisi huonoa. Toisaalta integroi-

dut kehitysympäristöt ovat mahdollistaneet aikaisempaa nopeamman ja tehokkaamman ohjelmistokehityksen. [4]

2.4 Integroitujen kehitysympäristöjen käyttö

Vuonna 2005 British Columbia yliopistossa tehtiin tutkimus, jossa tutkittiin Eclipse-alustan käyttöä. Käyttäjät käyttivät Eclipse-alustan versioita 3.1 ja 3.2. Tutkittavasta 99 henkilöistä 74 suostui datan luovuttamiseen. Data koostui jokaisen tutkittavan kehittäjän käyttöhistoriasta, joka oli kerätty Mylar Monitor -lisäosalla. Historiatiedot sisältävät informaatiota muutetuista asetuksista, ajetuista komennoista ja editorin valinnoista. Jokainen tallennettu tapahtuma sisältää tiedon sen ajasta, kuvauksen ja minkä lisäosan toiminto oli kyseessä. [6] Taulukossa 1 esitetään tutkittavien henkilöiden ammattien osuudet ja taulukossa 2 esitetään organisaatioiden koot, joissa he työskentelivät tutkimuksen aikana.

Taulukko 1: Ammatit [6]

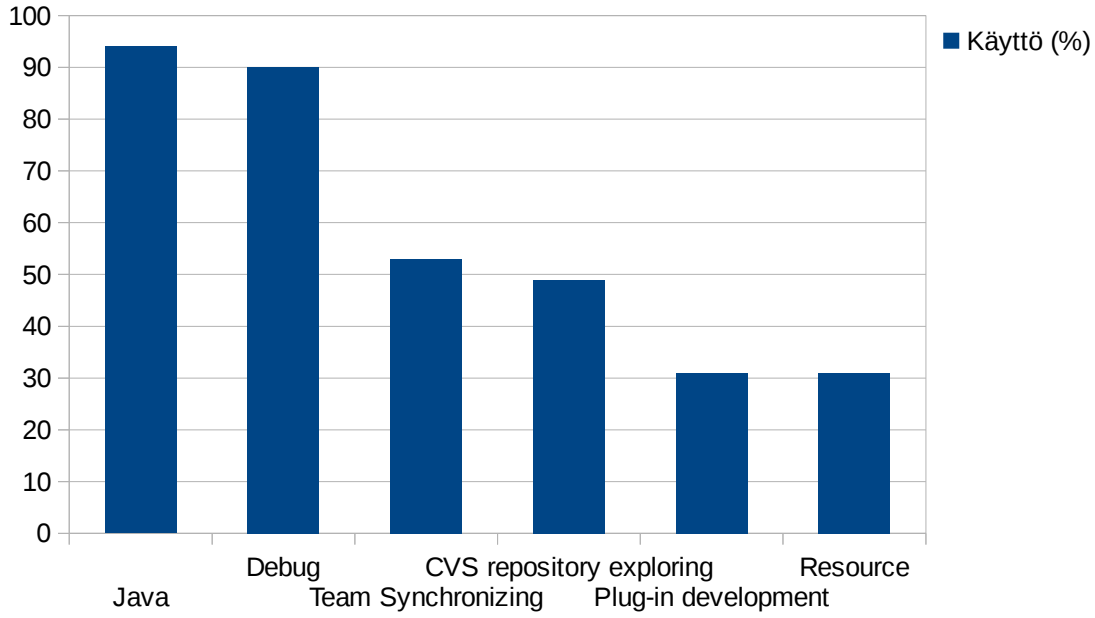
Ammatti	**Osuus tutkituista (%)**
Sovelluskehittäjä	65
Akateemikko	13
Sovellusarkkitehti	12
Johtaja	4
Muu	6

Taulukko 2: Organisaatioiden koot [6]

Organisaation koko	**Osuus tutkituista (%)**
Yksi henkilö	19
Alle 50	32
50-500 henkilöä	26
Yli 500 henkilöä	23

Suurin osa tutkituista oli sovelluskehittäjiä, ja tutkituista useimmat työskentelivät organisaatioissa, joissa työskenteli 2- 50 henkilöä (taulukot 1 ja 2).

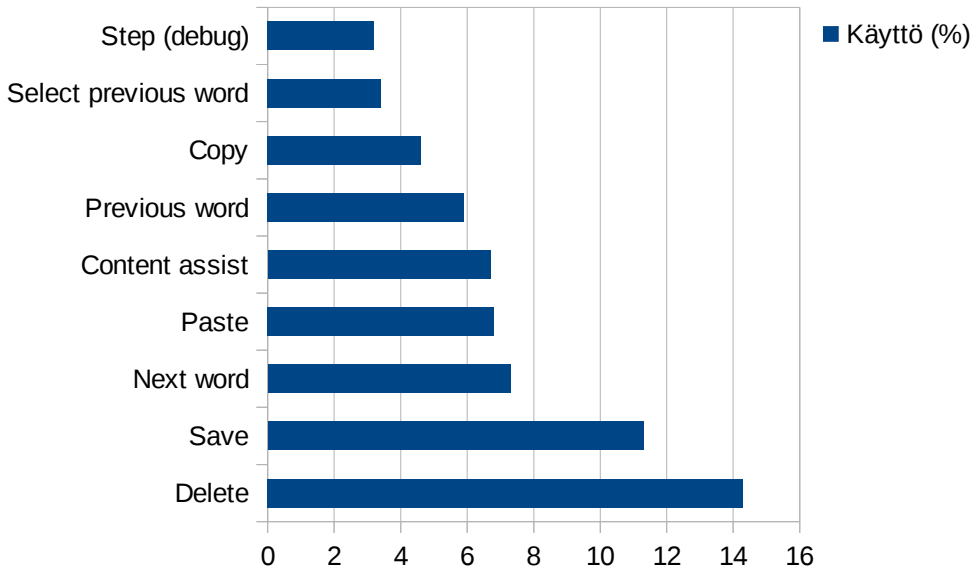
Eclipse mahdollistaa eri näkymien käytön. Näkymät muuttavat Eclipse-alustan ulkonäköä ja mahdollistavat täten työkohtaisemman ympäristön kuin, mitä voitaisiin saavuttaa geneerisellä näkymällä. Kuvassa 1 on esitetty Eclipse-alustan näkymät, joita ainakin 25 % kehittäjistä käyttivät.



Kuva 1: Näkymien käyttö Eclipsessä

Kuvasta 1 voidaan havaita, että Java- ja Debug-näkymät olivat kaksi suosituinta Eclipse-alustan näkymää. Tästä voidaan päätellä, että Java-ohjelmointi on yksi Eclipse-alustan suosituimmista käyttötarkoituksista. Debug-näkymän suosio viittaa siihen, että suuri osa ohjelmistokehitykseen käytetystä ajasta menee ohjelman virheiden etsintään. CVS-näkymän suosio viittaa tiimi- ja versionhallinta työkalujen yleisyyteen ohjelmistokehittäjien keskuudessa.

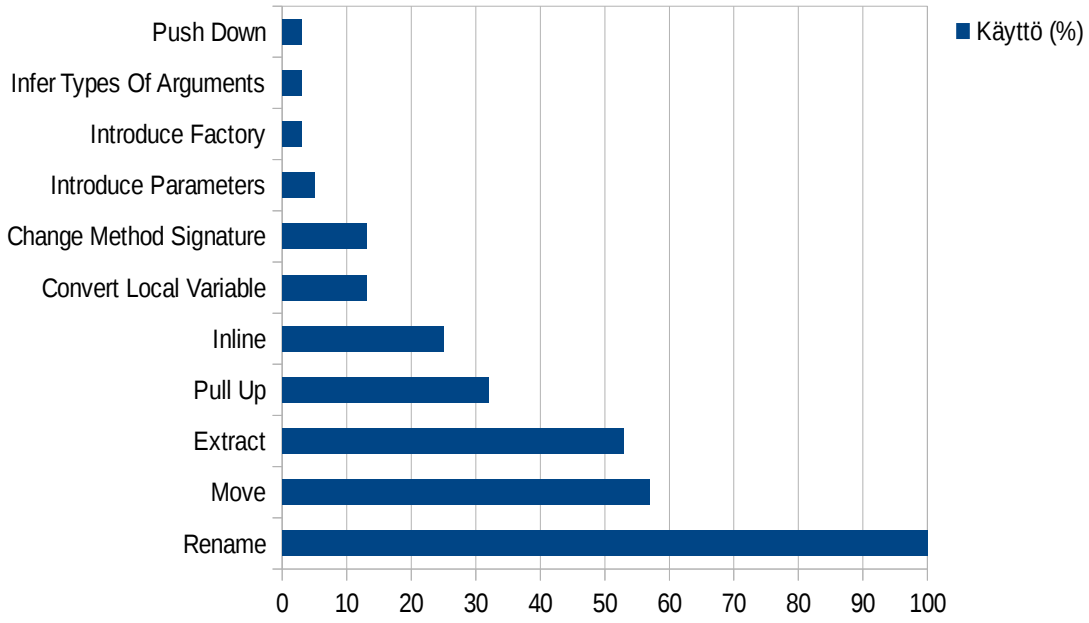
Kuva 2 esittää käytön mukaisesti kymmenen yleisintä komentoa. Pylväsdiagrammi kuvaa keskiarvokäyttöä kaikkien ohjelmistokehittäjien keskuudessa.



Kuva 2: Komentojen käyttö

Content assist -komennon käyttö on yllättävästi yhtä suosittua kuin normaalien tekstinmuokkausominaisuuksien käyttö, mikä on indikaattori automaattisen täydennyksen suosiosta.

Refaktoroinnilla tarkoitetaan lähdekoodin uudelleen muokkaamista sen laadullisten ominaisuuksien parantamiseksi ilman, että ohjelmiston toiminnallisuutta muutetaan. Esimerkiksi funktion nimi voidaan muokata paremmin kuvaavaksi. Kuvassa 3 on esitetty suosituimmat refaktorointikomennot Eclipse-alustassa.



Kuva 3: Refaktorointikomentojen käyttö

Nauhoitetut refaktorointikomennot on voitu ajaa sekä näppäin että käyttöliittymän valikoiden kautta. Komentoja, jotka refaktoroivat useampaa kuin yhtä luokkaa, käytettiin yleensä valikoiden kautta, kun taas yhtä luokkaa muokkaavat komennot ajettiin yleensä näppäinyhdistelmän avulla. [6]

3 SAMANKALTAISET OHJELMISTOT

Tämän luvun tarkoituksena on tutustua erilaisiin integroituihin kehitysympäristöihin ja SQF-lähdekoodieditoreihin. Tavoitteena on tehdä kilpailijatutkimusta ja löytää markkinarako, johon kehitettävä Side voi sijoittua. Hyväksi todettuja ratkaisuja voidaan parantaa tai siirtää sellaisenaan kehitettävään ohjelmaan hyödyntämällä esimerkiksi takaisinmallinnusmenetelmiä.

Tärkein askel kilpailuanalyysissä on luoda ulottuvuusjoukko, jonka kautta kilpailijoita voidaan verrata [15]. Tässä tilanteessa ulottavuusjoukko käsittää joukon integroitujen kehitysympäristöjen yleisiä ominaisuuksia. Ominaisuusjoukko pyritään muodostamaan tunnettujen integroitujen kehitysympäristöjen avulla. Tutkimalla olemassa olevia SQF-lähdekoodieditoreita voidaan todeta, että kaikkia yleisesti käytettyjä integroitujen kehitysympäristöjen ominaisuuksia ei niistä löydy.

Aluksi tutkitaan suosittuja integroituja kehitysympäristöjä. Luvun tavoitteena on muodostaa kuva siitä, miten menestyvä integroitu kehitysympäristö toimii. Tutkittavat suositut integroidut kehitysympäristöt ovat Qt Creator, Visual Studio, Eclipse-JDT ja NetBeans. Kaikki nämä ohjelmat ovat saavuttaneet merkittävän aseman ohjelmistokehityksen työkaluina. Toisessa vaiheessa tutkitaan SQF-lähdekoodieditoreita. Tavoitteena on selvittää niiden ominaisuudet ja niihin suhtautuminen. Lopuksi tehdään päätelmä, jonka perusteella havaitaan markkinarako Side-kehitysympäristölle. Erityisesti pyritään osoittamaan, että olemassa olevat SQF-lähdekoodieditorit eivät täytä menestyvän integroidun kehitysympäristön kriteereitä.

3.1 Suositut integroidut kehitysympäristöt

Tässä luvussa esitellään suosittuja integroituja kehitysympäristöjä. Tavoitteena on selvittää, miksi kyseiset ohjelmistot ovat saavuttaneet suuren suosion. Erityisen tärkeät ominaisuudet pyritään selvittämään, jotta niitä voidaan käyttää vaatimuslistauksen perusteina.

3.1.1 Qt Creator

Qt Creator on lisäosia tukeva integroitu kehitysympäristö Javascript, C++ [5], QML ja Qt [5] teknologioille. Qt Creator -kehitysympäristön lähdekoodieditorin ominaisuuksiin kuuluu syntaksin värjäys ja automaattinen täydennys [5]. Poikkeuksellisesti Qt Creator

ei tue välilehtiä. Qt Creator -kehitysympäristöön on integroitu Qt Designer -ominaisuudet graafisen käyttöliittymän suunnittelun mahdollistamiseksi [5]. Kehitysympäristö on näennäisesti alustariippumaton ja toimii sekä Linuxilla että Windowsilla [5]. Alustariippumattomuus on mahdollista, koska Qt-ohjelmistot ovat yleensä lähdekooditasolla alustariippumattomia. Qt Creator on osittain avointa lähdekoodia ja se on julkaistu LGPL-lisenssin alaisena.

3.1.2 Eclipse-JDT

Eclipse-JDT on Eclipse-alustan päälle rakennettu Java-ohjelmistokehitykseen tarkoitettu työkalukokoelma. Lyhenne JDT tulee sanoista Java development tools (suom. Javan kehitystyökalut). JDT on laajennettavissa sen ohjelmistorajapintojen kautta. JDT-lisäosat on jaettu alaryhmiin: JDT APT, JDT Core, JDT Debug, JDT Text, ja JDT UI. JDT APT tarjoaa tuen annotaatioiden prosessointiin. JDT Core huolehtii käyttöliittymän infrastruktuurista. Se tarjoaa tuen Java-elementtipuun (engl. Java element tree) navigointiin. Java-elementtipuu määrittelee Java-keskeisen näkymän projektista. Se kykenee esittämään elementtejä kuten paketteja, käännösyksiköitä (engl. compilation units), binääriluokkia, tyyppejä, metodeja ja kenttiä. JDT Core mahdollistaa myös hakutoiminnot. JDT Debug toteuttaa lisäosan virheidenetsintäominaisuudet. JDT Text toteuttaa Java-lähdekoodieditorin, johon sisältyy tuki syntaksikorotukselle, lähdekoodiavustukselle (Javadoc) ja metoditason editoinnille. JDT Text sisältää tuen reunailmoituksille, jotka ilmaisevat lähdekoodiongelmat, debug-pysähdyskohdat ja hakutulokset. JDT Text toteuttaa tuen lähdekoodin muotoilulle. JDT-UI toteuttaa Java-kohtaisen käyttöliittymän, johon sisältyvät pakettiselain, tyyppi-hierarkia-näkymä, Javan yleisnäkymä ja avustajat (engl. wizard) Java-elementtien muokkaamiseksi. JDT UI toteuttaa refaktorointituen, joka mahdollistaa esimerkiksi turvallisen Java-elementin uudelleen nimeämisen. Käyttäjä voi katselmoida mahdollisia refaktorointimuutoksia ennen niiden toteutumista. JDT Textin etsintätuen avulla hakuja voidaan rajata esimerkiksi paketteihin, tyyppeihin ja metodeihin. JDT Textin tarjoaman vertailutuen avulla voidaan vertailla rakenteellisia Java-käännösyksiköitä. Se näyttää eriäväisyydet Java-metodeissa ja tukee yksittäisten Java-elementtien korvaamista paikallisesta historiasta löytyvillä versioilla. [17]

Eclipse-JDT on näennäisesti alustariippumaton. Se toimii Windows-, Mac OS X -, Linux-käyttöjärjestelmissä sekä niiden 64 bittisissä varianteissa. Hyvä alustariippumattomuus on mahdollista, koska ohjelmisto on kehitetty Java-ohjelmointikielellä, jolloin ohjelmiston toiminnallisuus koostuu tavukoodista, joka voidaan ajaa kaikkien edellä mainittujen alustojen päällä. Eclipse-alustan [8] ja Eclipse-JDT:n lähdekoodi on avoin.

3.1.3 Visual Studio

Visual Studio on Microsoftin kehittämä integroitu kehitysympäristö, jonka pääasiallinen tarkoitus on helpottaa ohjelmien kehitystä Windows-käyttöjärjestelmälle. Sen tukemia ohjelmointikieliä ovat esimerkiksi C#, C++, ja C. Visual Studion ominaisuuksiin kuuluu tuki lähdekoodin automaattiselle täydennykselle IntelliSense-teknologian avulla, lähdekoodin refaktorointituki, integroitu virheidenetsintä, käyttöliittymäeditori ja tuki lisäosille. [33] Visual Studio toimii vain Windows-käyttöjärjestelmän päällä [25]. Visual Studion lähdekoodi on suljettua.

3.1.4 NetBeans

NetBeans on Java-ohjelmointikielellä toteutettu kehitysympäristö. Sen ensisijainen tarkoitus on olla työkalu Java-ohjelmien kehityksessä, mutta se tukee myös muita ohjelmointikieliä kuten PHP, C ja C++. Netbeans on alustariippumaton ja tukee Windowssia, Mac OS X:ää, Linuxia, Solarista ja monia muita käyttöjärjestelmiä. Alkujaan Netbeans oli Charles-yliopiston opiskelijaprojekti, mutta myöhemmin se on siirtynyt Oraclen omistukseen [29]. Netbeans-kehitysympäristön lähdekoodi on kuitenkin avoin [26]. Eclipse-alustan tapaan Netbeans on modulaarinen ja siitä on useita eri käyttökohdennettuja julkaisuja. Netbeans-alustaan integroituja moduuleita ovat esimerkiksi NetBeans profiler ja graafisen käyttöliittymän suunnittelutyökalut. NetBeans profilerin tarkoitus on avustaa ohjelmoijaa optimoinnissa, ja graafisen käyttöliittymän suunnittelutyökalut mahdollistavat Swing-käyttöliittymien suunnittelun graafisesti. [7]

3.2 SQF-editorit

Tämän alaluvun tavoitteena on esitellä olemassa olevia SQF-lähdekoodieditoreita. Tavoitteena on selvittää, miten Side kykenee kilpailemaan kyseisten ohjelmien kanssa. Lisäksi pyritään muodostamaan kuva SQF-kohtaisista ominaisuuksista, jotta tietoa voidaan käyttää vaatimusmäärittelyn pohjana. SQF-editoreiden suosion perusteena käytetään Armaholic-sivuston klikkausmittaria, joka mittaa editorikohtaisen internetsivun avauskertojen määrää. Tutkittavien editoreiden valintaperusteet ovat seuraavat: Notepad++ valittiin sen suosion perusteella; Squint valittiin sen syntaksin tarkistusominaisuuden takia; ArmaDev valittiin, koska se on Eclipse-lisäosa ja muistuttaa kaikista kolmesta eniten Side-kehitysympäristöä.

3.2.1 Notepad++ SQF Syntax Highlight and Auto Completion

Notepad++-editorin lisäosa on Armaholic-sivuston klikkausmittarin perusteella suosituin Arma 3 -pelin SQF-editori. Suosiostaan huolimatta se on todella yksinkertainen. Sen ominaisuuksia ovat:

- Syntaksikorotus
- Automaattinen täydennys [32]

Suuri osa SQF-ohjelmoijista käyttää kyseistä kieltä pienien komentosarjojen tekemiseen, joka voi omalta osaltaan selittää yksinkertaisuuden suosion. Notepad++-editorin SQF-lisäosa sisältää myös asetukset edellä mainittujen ominaisuuksien muokkaamiseksi [32].

3.2.2 Squint

Squint on vuonna 2010 julkaistu SQF-lähdekoodieditori ja staattinen analysaattori Arma 2 Operation Arrowheadin SQF-lähdekooditiedostoille. Sen ominaisuuksia ovat:

- Mahdollisuus muokata sqf-, sqm-, cpp-, ext- ja O2-tiedostoja
- Syntaksikorotus
- Syntaksin tarkistus
- Korjauksien ehdotus
- Mahdollisuus muokata PBO-arkistojen sisältämiä lähdekooditiedostoja suoraan.
- Tuki viitattujen tiedostojen tulkintaan ja niiden kautta tehtävään virheentarkistukseen. [16]

Squint on yksi harvoista ohjelmista, jonka avulla SQF-syntaksi voidaan tarkistaa pelin ulkopuolella. Suuresta ominaisuusmäärästä huolimatta Squint ei ole saavuttanut suurta suosiota. Osasyynä tähän voidaan pitää ohjelman merkittävää virheellisyyttä: ohjelma kaatuilee käytön aikana, ja SQF-syntaksin tarkistus ilmoittaa validin lähdekoodin virheelliseksi. Diplomityön kirjoitushetkellä Squint-kehitysympäristön bugienhallintasivustossa oli 71 avointa bugia [21]. Squint ei myöskään tue Arma 3 tai uusimpaa Arma 2: Operation Arrowhead SQF-versiota [16], mikä on vahva indikaattori sen kehityksen päättymisestä.

3.2.3 ArmaDev

ArmaDev on Eclipse-alustan päälle kehitetty lisäosa Arma-pelisarjaan liittyvien tiedostojen muokkaamiseen. ArmaDev-editorista puuttuu tuki Arma 3 -pelin SQF-versiolle. Se on julkaistu Creative Commons Attribution Unported -lisenssin alaisena. ArmDev tukee seuraavia ominaisuuksia:

- Syntaksin väritys: tunnukset väritetään niiden tyypin ja niiden hyväksyvien argumenttikombinaatioiden perusteella. Merkkijonot, kommentit ja muut elementit ovat myös väritettyjä.
- Automaattinen sisennys ja asettelu: Sulkeet sisennetään automaattisesti ja kulmasulkeet voidaan täydentää automaattisesti.

- Sisällön avustaja: Kursorin vienti varatun sanan tai tunnuksen päälle avaa näkymän, jossa on selitys sen toiminnasta, johon sisältyy hyväksytyt parametrit. Sivupalkissa voidaan näyttää kyseinen informaatio pysyvästi.
- Automaattinen täydennys: Painamalla CTRL+SPACE editori ehdottaa täydennystä kirjoitetulle lähdekoodille.
- Integroitu projektin hallinta: ArmaDev tukee kahta projektiavustajaa (engl. project wizard):
    1. Uusi tehtävä lähdekoodista: Luo uuden projektin valmiista tehtävästä.
    2. Vapaamuotoinen tehtävä: Luo uuden tehtävän tyhjästä.
- COMREF-näkymä: Näyttää kontekstiriippuvaisen komentonäkymän.
- Osittain toteutettu tuki tehtävämallille.
- RPT-lokin katselumahdollisuus [14]

ArmaDev muistuttaa suurelta osin Side-kehitysympäristöä. Tämä tarkoittaa sitä, että sen ominaisuuksien jäljittelyn lisäksi voidaan myös jäljitellä sen toteutustapaa hyödyntämällä takaisinmallinnusmenetelmiä kuten lähdekoodin takaisinkääntöä (engl. decompile). Side pyrkii kilpailemaan ArmaDev-ohjelmiston kanssa toiminnallisuuden määrän ja laadun avulla.

3.3 Käyttäjätutkimus SQF-editoreista

Tein käyttäjätutkimuksen arvioimalla SQF-editoreiden käyttäjien lähettämiä keskustelupalstaviestejä Armaholic-sivustolle. Jokaisesta Armaholic-sivustolle julkaistusta ohjelmasta on olemassa julkaisukeskustelu. Squint-kehitysympäristöllä on erillinen bugienhallintasivusto (engl. bug tracker), josta keräsin myös käyttäjätietoja. Erityistä huomioita kiinnitin puuttuvien ominaisuuksien toivomiseen, koska niiden toteutuksella voidaan saavuttaa kilpailuetua.

Notepad++ julkaisukeskusteluun oli diplomityön kirjoitushetkellä vastattu 110 viestillä. Johtuen Notepad++ SQF-lisäosan yksinkertaisuudesta, on ymmärrettävää, että myös ominaisuustoivomuksia oli vähän. Folding-tukea pyydettiin kaksi kertaa. CBA-funktioiden ja funktioviitteiden tukea pyydettiin kerran. [20]

Squint-kehitysympäristön julkaisukeskusteluun oli lähetetty diplomityön kirjoitushetkellä yhteensä 472 viestiä. Keskustelussa yleisin ominaisuuspyyntö oli Arma 3 -tuki, jota pyydettiin kolme kertaa. Seuraavaksi yleisin ominaisuuspyyntö oli Squint-kehitysympäristön lähdekoodin julkaisu, jota pyydettiin kaksi kertaa. [19] Toinen lähde Squint-kehitysympäristön käyttäjätutkimukselle oli sen bugienhallintasivusto, johon oli lähetetty muutamia ominaisuuspyyntöjä. Sekä julkaisukeskustelussa että virheiden hallintasivustolla toivottiin mahdollisuutta muuttaa tabulaattoripainalluksen ulostulo väleiksi. [19][21] Suurta julkaisukeskustelun viestimäärää selittää Squint-kehitysympäris-

tön bugien raportointi, joista suuri osa viesteistä koostui. Ottaen huomioon Squint-ohjelmiston julkaisukeskustelussa ja bugienhallintasivustolla olevien bugiraporttien määrä voidaan olettaa, että Squint on menettänyt merkittävän osan asiakaskunnastaan ohjelman virheellisen toiminnan takia.

ArmaDev-ohjelmiston julkaisukeskusteluun oli lähetetty yhteensä 37 viestiä. ArmaDev-ohjelmiston kohdalla havaitsin, että suosituin ominaisuustoivomus oli tuki Arma 3 -pelin funktioille. Yhteensä tätä ominaisuutta toivottiin neljä kertaa. Seuraavaksi eniten pyydettiin ohjelman lähdekoodin julkaisemista, mitä pyydettiin kaksi kertaa. Ominaisuuksia kuten syntaksin tarkistusta toivottiin myös. [18]

Analysoin SQF-editoreiden käyttäjiä ja havaitsin, että ominaisuuspyyntöjen välillä oli vähän toistuvuutta, minkä takia pelkästään niiden perusteella Side-kehitysympäristön arkkitehtuuripäätöksien tekeminen on kyseenalaista. Pidin saatua tietoa kuitenkin ohjeistuksena ja analysoin ominaisuustoivomuksien korrelaatiota muiden tutkimustuloksien kanssa. Esimerkiksi Squint-kehitysympäristöstä puuttuva tabulaattorin ulostulon muunnos välilyönneiksi löytyy kaikista tutkituista suosituista integroiduista kehitysympäristöistä, joten pidän siihen liittyvää toivomusta tarpeeksi hyvänä perusteluna ohjelmistovaatimukselle[1].

Havaitsin käyttäjätutkimuksen aikana, että Arma-yhteisö arvostaa avointa lähdekoodia, koska sen avulla voidaan varmistaa ohjelman pitkä elinkaari. Jokaisen ohjelman tuen loputtua löytyi vapaaehtoisia, jotka olivat valmiina jatkamaan projektia, mutta eivät aina kyenneet, koska ohjelman lähdekoodia ei ollut avoimesti tarjolla.

3.4 Päätelmät

Ominaisuuksien joukko, joka muodostettiin yleisluontoisten suosittujen integroitujen kehitysympäristöjen perusteella, on esitetty alapuolella. Ominaisuusjoukkoa voidaan pitää markkinointimielessä validina tutkittujen integroitujen kehitysympäristöjen suosion perusteella. Täten kyseisten ominaisuuksien sisällyttämistä kehitettävään integroituun kehitysympäristöön voidaan harkita.

1. **Refaktorointi**: Tällä tarkoitetaan lähdekoodin uudelleen muokkaamista sen laadullisten ominaisuuksien parantamiseksi ilman, että ohjelman toiminnallisuutta muutetaan. Kaikista käsitellyistä tunnetuista integroidusta kehitysympäristöstä löytyy ainakin ominaisuus elementtien turvalliselle uudelleen nimeämiselle.
2. **Automaattinen täydennys**: Tällä tarkoitetaan ominaisuutta, jossa integroitu kehitysympäristö ehdottaa mahdollista täydennystä kirjoitetulle lähdekoodille. Esimerkiksi funktion nimi voidaan täydentää loppuun automaattisen täydennyksen avulla. Kaikista käsitellyistä tunnetuista integroiduista kehitysympäristöistä löytyy tämä ominaisuus.

1 Eclipse tukee jo valmiiksi kyseistä ominaisuutta, joten sitä ei tarvitse erikseen toteuttaa.

3. **Sisällön avustaja**: Sisällön avustajalla tarkoitetaan ominaisuutta, minkä avulla ohjelmoija voi saada lisätietoa käyttämistään ohjelmistokirjastoista. Esimerkiksi Qt Creator -kehitysympäristössä tämä tapahtuu painamalla F1-pikanäppäintä, kun kursori on jonkin Qt-entiteetin päällä.
4. **Syntaksikorotus**: Syntaksikorotuksella tarkoitetaan lähdekoodieditorin ominaisuutta, joka värittää tunnukset niiden tyypin perusteella. Syntaksikorotus parantaa lähdekoodin luettavuutta ja auttaa ohjelmoijaa välttämään syntaktisia virheitä.
5. **Syntaksin tarkistus**: Syntaksin tarkistaja ilmoittaa lähdekoodissa olevista syntaktisista virheistä lähdekoodin kirjoitushetkellä. Syntaksin tarkistus ja maalaus helpottavat yhdessä virheiden löytämistä ja auttavat välttämään niitä.
6. **Korjauksen ehdotus**: Korjauksen ehdotus on ominaisuus, joka ehdottaa ohjelmoijalle mahdollista syntaktisen virheen korjaustapaa. Korjauksen ehdotus nopeuttaa virheen korjaamista.
7. **Virheidenetsintätuki**: Tämä ominaisuus auttaa ohjelmoijaa löytämään semanttisia ohjelmointivirheitä. Ominaisuus tarkoittaa muun muassa mahdollisuutta ilmoittaa, missä lähdekoodin rivillä ohjelma kaatui ajonaikaisesti.
8. **Käyttöliittymäeditori**: Tällä tarkoitetaan graafista editoria, joka mahdollistaa käyttöliittymän piirtämisen integroidun kehitysympäristön sisällä. Ominaisuus nopeuttaa käyttöliittymän toteutusta, koska sen ulkonäkö nähdään toteutuksen aikana. Yleensä kyseinen ominaisuus sisältää myös käyttöliittymän toiminnallisuuteen liittyviä lähdekoodigeneraattoreita.
9. **Laajennettavuus**: Tällä ominaisuudella tarkoitetaan mahdollisuutta, että integroitua kehitysympäristöä voidaan laajentaa erinäisillä lisäosilla. Lisäosatuki pidentää ohjelman elinkaarta, koska kehitykseen voi helposti osallistua kolmansia osapuolia.
10. **Ohjelman ajaminen**: Tällä ominaisuudella tarkoitetaan, että kehitettävä ohjelma voidaan käynnistää integroidusta kehitysympäristöstä käsin.

Edelle esitetyistä ominaisuuksista kaikki paitsi ominaisuus kuusi sisältyivät suosittuihin integroituihin kehitysympäristöihin. Hyödyntämällä luvun kaksi käyttäjätutkimustietoja voidaan havaita, että tämän tyyppisille ominaisuuksille on yleisen ohjelmistokehityksen alalla myös käyttöä. SQF-projekteista suurimmat voivat olla kooltaan kymmeniä tuhansia rivejä, missä tilanteessa eroavaisuudet perinteiseen ohjelmistokehitykseen ovat vähäisiä. Side pyrkii olemaan integroitu kehitysympäristö tällaisille projekteille.

Taulukossa 3 on esitetty olemassa olevat SQF-editorit ja niiden toteuttamat yleisluontoiset integroitujen kehitysympäristöjen ominaisuudet. RPT-lokituki mielletään osaksi virheidenetsintää.

Taulukko 3: Toteutetut ominaisuudet

	Ohjelma		
Ominaisuus	Notepad++	Squint	ArmaDev
1. Refaktorointi	X		X
2. Automaattinen täydennys	X	X	X
3. Sisällön avustaja		X	X
4. Syntaksikorotus	X	X	X
5. Syntaksin tarkistus		X	
6. Korjauksen ehdotus		X	
7. Virheidenetsintätuki			X
8. Käyttöliittymäeditori			
9. Laajennettavuus			
10. Ohjelman ajaminen			

Edellä esitystä ominaisuuksista 1, 2 ja 4 on sisällytetty Notepad++ SQF-lisäosaan. Ominaisuudet 2-6 löytyvät Squint-kehitysympäristöstä. Ominaisuudet 1, 2, 3, 4 ja 7 löytyvät Armadev-Eclipse-lisäosasta. Voidaan havaita, että mikään käsitellyistä SQF-editoreista ei toteuttanut kaikkia ominaisuuksia. Ominaisuuksia, joita mikään SQF-editori ei toteuttanut olivat: virheidenetsintätuki, käyttöliittymäeditori, laajennettavuus ja ohjelman ajaminen. Näiden ominaisuuksien toteuttamisella voidaan täten saavuttaa kilpailuetua, mistä johtuen on perusteltua olettaa, että Side-kehitysympäristölle on markkinarako olemassa.

4 MODAUS JA ARMA 3

Tässä luvussa käsitellään pelien yhteisöpohjaista muokkaamista eli modausta (engl. modding). Aluksi kerrotaan modaamisesta yleisellä tasolla ja sen jälkeen kerrotaan, miten modaaminen liittyy Arma 3 -peliin. Erityistä huomioita kiinnitetään SQF-ohjelmointikielen rooliin Arma 3 -pelin modauksessa.

4.1 Modaus

Osasta pelejä ja pelimoottoreita löytyy tuki ohjelmointikielelle, joka mahdollistaa toiminnallisuuden kehittämisen yhteisön tai pelin kehitystiimin toimesta. Osa näistä peleistä tukee toimialakohtaista ohjelmointikieltä, josta esimerkkinä Bohemia Interactive Studion pelien SQF. Toisaalta osa peleistä hyödyntää jo olemassa olevia ohjelmointikieliä, esimerkiksi Battlefield 2 käyttää Python-komentosarjakieltä ja tarjoaa ohjelmistorajapinnan pelin kanssa kommunikoimiseen. Riippuen kielen tai ohjelmistorajapinnan ominaisuuksista ohjelmoija voi kehittää hyvinkin laajalla skaalalla toiminnallisuutta pelin sisälle. Kehitetyt lisäosat pyrkivät muokkaamaan enemmän tai vähemmän peliä ja niistä käytetään termiä modi. Modi voi tarkoittaa mitä tahansa peliin tehtyä muokkausta, eikä siis rajoitu vain ohjelmoinnilla toteutettuun toiminnallisuuteen. Modin tarkoitus voi olla esimerkiksi pieni muutos pelin fysiikoissa tai täydellinen muokkaus, jolloin pelin tarina tai tyyppi ei ole enää alkuperäistä vastaava. [11] Arma 3 -pelin tehtävät kykenevät täyttämään yleisluontoisen modimääritelmän, mutta selkeyden takia niistä käytetään termiä ”tehtävä”. Arma 3 -modi tarkoittaa tässä diplomityössä lisäosaa, jonka käyttöönotto tapahtuu *mods*-käynnistysparametrin avulla.

Moditiimillä tarkoitetaan modia kehittävää tiimiä. Tiimin koko voi erota muutamasta henkilöstä muutamaan kymmeneen henkilöön. Esimerkiksi Battlefield 1942 -pelille kehitetyn Home Front -modin kehitystiimiin kuului aikoinaan 27 jäsentä. Moditiimien kokoonpano koostuu suurelta osin samantyyppisestä roolituksesta kuin pelikehitystiimienkin. Moditiimiin voi kuulua esimerkiksi teksturoijia (engl. texture artist), ääninäyttelijöitä, 3D-mallintajia ja ohjelmoijia. [11] Samankaltaisuuksista johtuen ei ole ennenkuulumatonta, että moditiimi siirtyykin kehittämään omaa peliään.

4.2 Arma 3

Arma 3 on diplomityön tekohetkellä Bohemia Interactive Studion kehittämän Arma-pelisarjan viimeisin julkaisu. Peli pyrkii mahdollisimman realistisesti simuloimaan laajamittaista sotatilannetta. Pelissä pelaaja osallistuu tehtäviin, joissa 2-4 osapuolta taistelevat keskenään. Osapuolet voivat osittain tai kokonaan koostua joko ihmis- tai tekoälypelaajista. Peli sisältää valmiina joitakin tehtäviä, mutta pelaajille annetaan myös mahdollisuus kehittää omia tehtäviään tehtäväeditorin avulla. Tilanteissa joissa tehtäviin halutaan monimutkaista toiminallisuutta, niihin voidaan sisällyttää SQF-lähdekoodia. Tehtävät voivat modien (luku 4.1) tapaan muokata peliä erittäinkin laajamittaisesti, esimerkiksi muuttaen pelin tarkoitusta, tavoitteita tai tarinaa.

Bohemia Interactive Studion pelit ovat tunnettuja niiden avoimesta muokattavuudestaan. Arma 3 -peliä voidaan laajentaa tehtävien ja modien avulla. Ainoastaan modit voivat sisältää dynaamisia kirjastoja, karttoja ja 3D-objekteja. Modit ja tehtävät voivat kuitenkin sisältää saman tasoisia SQF-muokkauksia. Pelin käynnistäessä modit määritellään *mods*-käynnistysparametrin avulla. Pelin yhdistäessä pelipalvelimelle varmistetaan, että pelillä on kaikki palvelimen vaatimat modit käytössä. Tehtävän käynnistyessä varmistetaan, että tehtävä löytyy peliltä, ja jos sitä ei löydy, se ladataan palvelimelta. Pelaajalta ei siis edellytetä manuaalisia toimia tehtävän lataamiseksi, mistä johtuen SQF-ohjelmoijat joskus suosivat muokkauksien sisällyttämistä tehtävään eivätkä modiin.

5 SQF

Status Quo Function (lyh. SQF) on Bohemia Intervactive Studion kehittämien pelien laajentamiseen tarkoitettu ohjelmointikieli. Se on Status Quo Scriptin (lyh. SQS) seuraaja. SQF on tarkoitettu lähinnä yhteisön käyttöön, mutta se on myös laajassa käytössä yrityksen sisällä. Arma-pelisarjassa tämä tarkoittaa tehtävien ja modien toiminnallisuuden laajentamista tekstuaalisella ohjelmoimilla. SQF-ohjelmointikielestä on useita pelikohtaisia variaatiota. Tässä diplomityössä keskitytään erityisesti Arma 3 -pelin SQF-variaatioon.

Tässä luvussa esitellään SQF-ohjelmointikielen perusteet. SQF-ohjelman perusrakenteet ovat muuttujat, komennot ja funktiot. Lisäksi SQF-ohjelmasta voidaan kutsua käyttöjärjestelmäkohtaista dynaamista kirjastoa, joka voi olla toteutettu esimerkiksi C++-ohjelmointikielellä.

5.1 Komennot

SQF-komennolla tarkoitetaan ohjelmointikielen varattua sanaa, joka tekee kutsuttaessaan toimenpiteen. Komento on verrattavissa muiden ohjelmointikielien funktioon tai proseduuriin, mutta SQF-ohjelmointikielessä funktio-termillä on muuta tarkoittava merkitys. Komentoja ei ole toteutettu SQF-ohjelmointikielellä eikä niiden toteutus ole täten avoin SQF-ohjelmoijalle.

SQF-komentoa kutsutaan parametrien kanssa tai ilman parametreja. Jos SQF-komento hyväksyy vain yhden parametrin, se esitetään komennon jälkeen, esimerkiksi: ”getPos player”. Tilanteissa, joissa SQF-komento hyväksyy useamman kuin yhden parametrin, on joissakin tapauksissa mahdollista esittää komennon ensimmäinen parametri komentokutsun edessä: ”parametri1 komento parametri2”. Kutsutapaa käytetään yleensä olio-ohjelmointityyppisesti, jolloin kutsutapa jäljittelee ulkonäöllisesti olion jäsenfunktion kutsua, esimerkiksi: ”player setPos _position”.

5.2 Muuttujat

SQF-ohjelmointikieli ei tue vahvaa tyypitystä, vaan muuttujan tyyppi vaihtuu sen arvon perusteella myös silloin, kun muuttujan edellinen arvo ylikirjoitetaan. SQF tukee monia ohjelmointikielille tavanomaisia tietotyyppejä kuten:

- Numero
- Merkkijono
- Liukuluku
- Olio
- Taulukko
- Totuusarvo

Tämän lisäksi SQF-ohjelmointikielessä on lukuisia pelikohtaisempia tietotyyppejä kuten *Side*, *Task* ja *Location*. Numero ilmaisee kokonaisluvun. Sen maksimi ja minimi on rajoitettu. Liukuluvulla ilmaistaan sekä positiivisia että negatiivisia reaalilukuja. Sen tarkkuus on rajoitettu. Merkkijonolla ilmaistaan tekstiä ja sen pituus on rajoitettu. Oliotyyppinen muuttuja viittaa johonkin pelin olioon kuten taloon tai pelaajaan. Taulukko on järjestetty kokoelma eri tyyppisiä muuttujia. Taulukossa muuttujien tyyppien kombinaatiolla ei ole merkitystä. Esimerkiksi yksi taulukko voi sisältää merkkijono- ja numerotyyppisiä muuttujia. Taulukko-tietotyyppi on olennaisessa osassa SQF-ohjelmointia, koska se on yksi harvoista tavoista yhdistää monen tyyppistä dataa yhden muuttujan alle. SQF-lähdekoodissa muuttujan näkyvyys voi vaihdella kolmella eri tavalla. Nämä tavat ovat:

1. Lokaali (engl. local)
2. Globaali (engl. global)
3. Julkinen (engl. public)

Lokaali muuttuja näkyy vain lohkon sisällä. Globaali muuttuja näkyy asiakasohjelman (pelin) tai pelipalvelimen koko ajoympäristössä. Julkinen muuttuja on kaikesta kolmesta näkyvin. Se on globaali muuttuja, joka lähetetään erillisellä *publicVariable*-komennolla kaikille tai osalle palvelimeen yhdistyneistä osapuolista. Aitoa automaattisesti päivittyvää julkista muuttujaa ei ole olemassa SQF-ohjelmointikielessä.

5.3 Funktiot

SQF-funktioita emuloidaan tallentamalla lähdekoodia muuttujaan, joka voidaan kutsua erillisellä komennolla:

```
_muuttuja = {koodi};
```

Funktiolla on samat näkyvyysalueet kuin muuttujalla, jotka ovat globaali, lokaali tai julkinen. Funktio voidaan myös esitellä asetustiedostossa, jolloin sen nimi on muotoa: ”<tägi>_fnc_<nimi>”. Funktion tägi ilmoitetaan asetustiedossa erikseen tag-ominaisuuden arvolla tai vaihtoehtoisesti tägi määrittyy asetusluokan perusteella. SQF-funktiot hyväksyvät vain yhden parametrin, mistä johtuen parametri on yleensä taulukko, joka sisältää useamman arvon. Funktio voidaan joko suorittaa erillisessä säikeessä *spawn*-komennolla tai samassa säikeessä *call*-komennolla. Alla olevassa lähdekoodiesimerkissä on esitetty tyypilliset funktion kutsutavat.

```
[parametri1, parametri2...] call tag_fnc_funktio;
[parametri1, parametri2..] spawn tag_fnc_funktio;
```

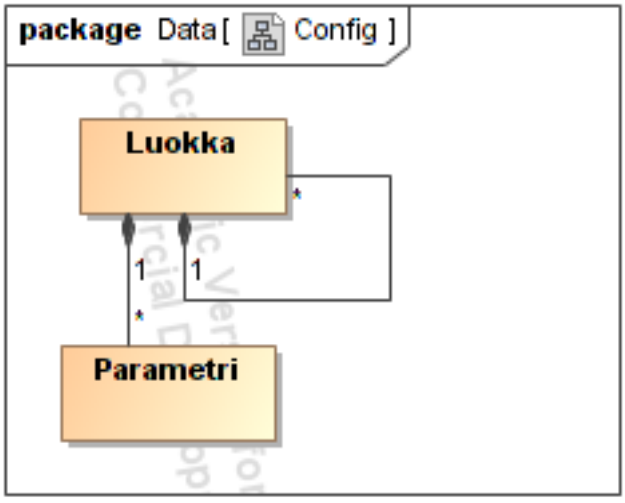
Muuttujana määritellyt funktiot kutsutaan samantyyppisesti, mutta funktiosta ilmoitetaan vain sen nimi. On tietenkin mahdollista nimetä muuttuja niin, että se sisältää myös ”tägi_fnc_”-etuliitteen.

```
[parametri1, parametri2..] call funktio;
```

Funktion kutsutapa vaikuttaa suoritettavan funktiokoodin mahdolliseen toiminnallisuuteen. Esimerkiksi eräässä SQF-versiossa *call*-kutsun käyttö estää *sleep*-komennon käytön funktiossa. Integroitu kehitysympäristö kykenee ilmoittamaan tämän tyyppisistä semanttisista ohjelmointivirheistä lähdekoodin tulkinnan avulla.

5.4 Tiedostot

SQF-ohjelmointiin liittyviä tiedostoja voidaan sisällyttää joko pelin tehtävään tai modiin. Näitä tiedostoja ovat asetustiedostot ja SQF-lähdekooditiedostot. SQF-lähdekooditiedostot määrittelevät Arma 3 -pelin sisällä toimivan ohjelman toiminnallisuuden. SQF-ohjelmoinnissa asetustiedostot määrittelevät luokkia varatulla sanalla *class*. Luokilla määritellään asetuksien osia ja niihin voidaan viitata SQF-lähdekoodista. Luokilla voidaan esimerkiksi määritellä tehtävien parametreja, käyttöliittymiä ja esitellä julkisia funktioita. Asetustiedostot muistuttavat syntaksiltaan C++-ohjelmointikielen otsikkotiedostoja ja käyttävät usein samoja tiedostopäätteitä *h* ja *hpp*. Kuvassa 4 on esitetty kaavio asetustiedoston rakenteesta.



Kuva 4: Asetustiedoston rakenne

Asetustiedosto koostuu luokista, jotka voivat sisältää muita luokkia. Luokka voi myös sisältää rajoittamattoman määrän parametreja. Parametrit voivat olla tyypiltään kokonaislukuja, merkkijonoja tai taulukoita. Jäsenmuuttujan tyyppi määrittyy SQF-lähdekoodin tapaan sijoitetun arvon perusteella.

5.5 Dynaamiset kirjastot

Pääasiallinen kolmannen osapuolen ohjelmallisen toiminnallisuuden lisääminen Arma 3 -peliin tapahtuu SQF-ohjelmoinnin kautta. Toimialakohtaisen kielen (engl. domain specific language) etuna nähdään tietoturvallisuus, koska toiminnallisuus rajoittuu pelin sisäiseen toimintaan. On kuitenkin tilanteita, joissa toiminnallisuuden halutaan vaikuttavan pelin ulkopuoliseen toimintaan. Tämän takia Arma 3 -peliin on lisätty tuki Windows- ja Linux-käyttöjärjestelmien dynaamisille kirjastoille. Arma 3 -peliä varten toteutetun dynaamisen kirjaston tulee sisältää *RVExtension*-funktion määritelmä. Windows-käyttöjärjestelmässä syntaksi on:

```
void __stdcall RVExtension(char *output, int outputSize, const char *function)
```

ja Linuxisssa syntaksi on:

```
void RVExtension(char *output, int outputSize, const char *function)
```

Funktiossa *output*-merkkijono on pelille lähetettävä viesti. *outputSize* kertoo *output*-merkkijonon maksimipituuden ja *function* on peliltä saatava viesti dynaamiselle kirjastolle. *function*-muuttujan koko on rajoitettu. SQF-ohjelmointikielen *callExtension*-komennolla voidaan kutsua *RVExtension*-funktiota pelistä käsin seuraavalla syntaksilla:

```
"<kirjaston nimi>" callExtension <function>
```

missä *function* on merkkijono. Dynaamisten kirjastojen käytössä on tärkeää ottaa huomioon, että kirjasto ei voi suoraan kutsua peliä, mutta peli voi kutsua kirjastoa.

6 ECLIPSE

Eclipse tarkoittaa kahta asiaa. Ensinnäkin Eclipse on avoimen lähdekoodin yhteisö Java perusteisten työkalujen kehittämiseen. Kaikista ilmiselvin Eclipse-yhteisön tuotos on Eclipse-alusta, joka on termin toinen merkitys. [8] Alustan päälle on kehitetty useita työkaluja aina C-ohjelmoinnista Web-kehitykseen asti [8], joista ehkä tunnetuin on Java-kehitykseen tarkoitettu JDT.

6.1 Eclipse RCP

Eclipse RCP (engl. Rich Client Platform) tarkoittaa Eclipse-alustaa, jonka avulla voidaan rakentaa rich client -tyyppisiä ohjelmistoja. Rich client tunnetaan myös nimellä fat client. Termi tarkoittaa ohjelmaa, jossa suuri osa koko ohjelmiston toiminnallisuudesta toteutetaan asiakasohjelman puolella. Kontrastina thin client voi esimerkiksi tarkoittaa web-sivua, jossa suuri osa toiminnallisuudesta sijaitsee palvelimen puolella. Thin clientin etuina ovat helppo ylläpidettävyys ja rich clientin etuna on pienet vasteajat. Eclipse RCP-alustan yleisin sovelluskohde on erinäköisten ohjelmistokehitykseen liittyvien lisäosien kehitys, mutta sen päälle on rakennettu myös muun muassa CAD-sovelluksia. Eclipse RCP on hyvin laaja, mikä antaa sen käyttäjälle laajat sovellusmahdollisuudet, mutta toisaalta käyttö on monimutkaista. Eclipse RCP hyödyntää laajennettavuudessa OSGi-kehystä.[8]

6.2 OSGi

Eclipse-alustaa voidaan laajentaa lisäosien avulla, joista käytetään usein termiä *bundle.* OSGi tarjoaa kehyksen näiden lisäosien kehittämiseen ja liittämiseen Eclipse-alustan päälle [8]. OSGi kykenee käyttämään laiskaa aktivointitapaa, joka tarkoittaa sitä, että lisäosa aktivoidaan vain ja kun siihen sisältyvää luokkaa käytetään [8]. Eclipse koostuu pääasiassa toisiinsa liitetyistä lisäosista [8], minkä takia OSGi-kehyksen voidaan ajatella luovan sen päälle ikään kuin komponenttialustan.

OSGi toimii hallitsemalla bundlejen luokkien latausta. Hallinta tapahtuu tulkitsemalla *MANIFEST.MF*-tiedostoa. Taulukossa 4 on esitetty *MANIFEST.MF*-tiedoston sisältämät kentät ja niiden selitykset.

Taulukko 4: MANIFEST.MF sisältö

Kenttä	Selitys
Bundle-Name	Bundlen nimi
Bundle-SymbolicName	Bundlen symbolinen nimi, jolla siihen voidaan viitata lähdekoodissa ja määrittelyissä.
Bundle-Version	Bundlen versio
Bundle-Activator	Luokka, jonka objekti luodaan bundlea ladatessa.
Bundle-Vendor	Bundlen tekijä
Require-Bundle	Bundlet, jotka pitää olla aktivoituna ennen tämän bundlen lataamista.
Bundle-RequiredExecutionEnverionment	Vaadittu Java-tulkki
Bundle-ActivationPolicy	Ainut validi arvo: ”lazy”, joka ottaa käyttöön laiskan aktivointitavan.
Export-Package	Määrittelee tarjotut Java-paketit tilanteessa, jossa bundle tarjoaa paketteja muiden bundlejen käyttöön.

MANIFEST.MF-tiedostossa on määritelty bundlen ominaisuudet ja sen riippuvuussuhteet muihin bundleihin. Bundlet otetaan käyttöön dynaamisesti riippuvuussuhteiden ja niiden tarpeen perusteella. [8]

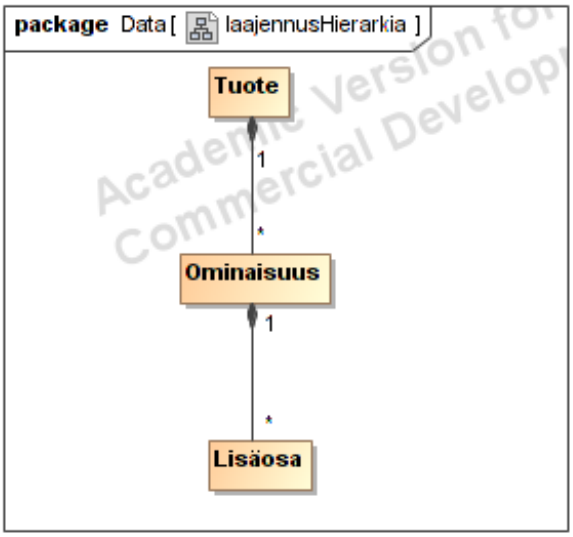
6.3 Laajennusrekisteri

Eclipse-alustan laajennusrekisteri (engl. extension registry) avulla voidaan määritellä suhteita erillisten Eclipse-alustan laajennuksien (lisäosien ts. bundlejen) välille. Eclipse-alustan lisäosat voivat määritellä itselleen laajennuspisteitä (engl. extension points), joiden kautta muut lisäosat voivat kommunikoida niiden kanssa. Laajennuspisteet ja niiden käyttö määritellään *plugin.xml*-tiedossa. Jokaisella Eclipse-lisäosalla on kyseinen tiedosto. Laajennusrekisterin periaatteena on laiska määrittely tarkoittaen sitä, että laajennuspisteeseen liitetty toiminnallisuus ladataan vasta silloin, jos tai kun sitä tarvitaan. [8]

6.4 Eclipse-alustan laajennushierarkia

Eclipse-alustan päälle rakennettu ohjelmisto voi olla joko laajennus tai erillinen ohjelmisto eli tuote (engl. product). Laajennus asennetaan olemassa olevan Eclipse-asennuk-

sen päälle ja erillinen ohjelmisto on tuote, joka itsessään sisältää Eclipse-alustan. Eclipse-alustan laajennushierarkia koostuu kolmesta kerroksesta. Ylimpänä on tuote, keskellä ovat ominaisuudet (engl. feature) ja alimpana lisäosat (engl. plug-in). Tuote voidaan määritellä myös koostuvaksi pelkistä lisäosista. Kuvassa 5 on esitetty kaavio Eclipse-alustan laajennushierarkiasta.



Kuva 5: Eclipsen laajennushierarkia

Yksittäinen tuote koostuu useista ominaisuuksista. Ominaisuus on kokoelma Eclipse-lisäosia. [8] Kolmitasoinen Eclipse-alustan laajennushierarkia mahdollistaa toteutettujen ominaisuuksien monenlaisen uudelleenkäytettävyyden.

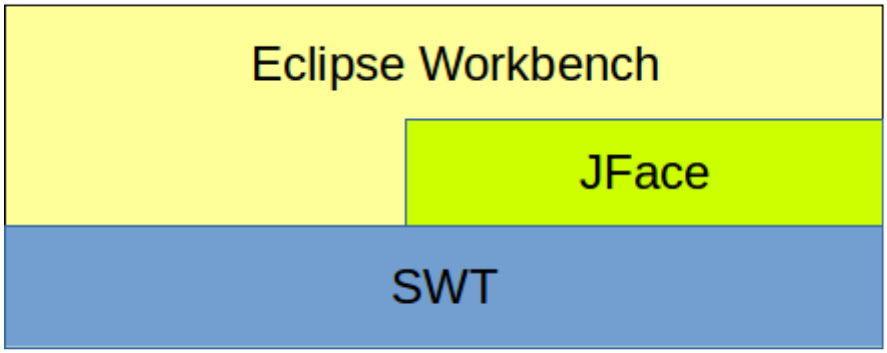
6.5 Eclipse-alustan käyttöliittymäkehykset

Eclipse-alustan päälle voidaan rakentaa käyttöliittymä kahden eri kehyksen avulla. Nämä ovat alemman tason SWT (engl. The Standard Widget Toolkit) ja ylemmän tason JFace. Eclipse-alustan päälle päätettiin kehittää SWT-kehys, kun jo olemassa olevat Java-käyttöliittymäkehykset AWT ja Swing osoittautuivat riittämättömiksi. Uuteen SWT-kehykseen lainattiin ideoita VisualAgesta[2] ja SmallTalkista[3]. SWT-kehykseen pyrittiin sisällyttämään AWT-kehyksen tapaan käyttöjärjestelmälle natiivien widgettien tuki. Tilanteessa, jossa natiiveja widgettejä ei ole tarjolla, käytetään Swing-kehyksen widgettejä. AWT pyrkii näin ollen yhdistämään Swing- ja SWT-kehyksien parhaat puolet, mikä varmistaa graafisen käyttöliittymän tehokkuuden ja ulkonäön. [23]

SWT julkaistiin vuonna 2001 osana Eclipse IDE:tä. Tämän julkaisun jälkeen SWT on kehittynyt omaksi julkaisukseen. SWT tukee useita käyttöjärjestelmiä, joihin sisältyvät Windows, Mac OS X ja Linux. SWT on avointa lähdekoodia ja se on julkaistu

2 IBM-yrityksen kehittämä integroitu kehitysympäristö.
3 Ohjelmointikieli, joka oli merkittävässä osassa MVC-suunnittelumallin yleistymisessä.

EPL-lisenssin (engl. Eclipse Public License) alla. [23] Kuvassa 6 on esitetty, miten JFace ja SWT asettuvat Eclipse-alustaan. Molempia sekä SWT-kehystä että JFace-kehystä käytetään osana Eclipse Workbenchiä.



Kuva 6: Eclipsen käyttöliittymäkehykset

JFace on abstraktiotasolla korkeammalla kuin SWT. JFace on SWT-kehyksen päälle rakennettu MVC-suunnittelumallin[4] mukainen grafiikkakehys. JFace ei piilota allensa SWT-kehystä, vaan sen käyttäminen onnistuu JFace-kehyksen läpi. [23]

4 MVC-suunnittelumallin mukaisesti ohjelmisto voidaan jakaa kolmeen osaan: malli (engl. model), näkymä (engl. view) ja käsittelijä (engl. controller). Malli on tietojen ohjelmallinen käsittelijä, näkymä piirtää tiedot ja käsittelijä käsittelee käyttäjän antamat käskyt.

7 XTEXT

Xtext on Eclipse-alustan päällä toimiva kehys, jonka avulla voidaan kehittää uusia ohjelmointikieliä tai editoreita olemassa oleville ohjelmointikielille. Xtext helpottaa ohjelmointikielen tuen kehittämistä Eclipse-alustan päälle, koska syntaksin tarkistajan ohjelmoinnin sijasta Eclipse-lisäosan kehittäjä määrittelee ohjelmointikielen syntaksin, minkä perusteella Xtext rakentaa editorin toiminnallisuuden. [9]

Xtext-kehyksen generoima editori tukee toimintoja kuten resurssienhallintaa ja lähdekoodin tulkintaa. Kielen määrittely tapahtuu erillisellä merkkikielellä nimeltään Grammar (suom. kielioppi). Grammar-kielen ideana on määritellä tarkasti ohjelmointikielen syntaksi ja sen semanttinen datamalli. Kielioppimäärittelyjen avulla Xtext muodostaa abstraktin syntaksipuun määrittelyllä kielellä kirjoitetusta lähdekoodista. Xtext tukee myös ohjelmointikielen semanttisuuden mallintamista, mutta diplomityössä keskitytään vain syntaktisuuden mallintamiseen.

7.1 Abstrakti syntaksipuu

Abstraktilla syntaksipuulla (engl, abstract syntax tree, lyh. AST) kuvataan syntaktista rakennetta, esimerkiksi lähdekoodia, puurakenteena. Se ei ota kantaa, miten kyseinen syntaksi käytännössä kirjoitetaan, mistä johtuen se mielletään abstraktina. Abstraktissa syntaksipuussa ohjelmointikielen eri rakenteet jaetaan osiin ja niiden välisiä koosteita kuvataan. Ohjelmointikielen määrittelyistä osa on primitiivistä, eli niitä ei voida jakaa osiin, kun taas osa muodostuu koosteista. [24] Esimerkiksi ehtolauseke voi koostua testistä, else-haarasta ja then-haarasta. Vastaavasti kokonaisluvun literaali 0 on primitiivinen, koska sitä ei voi jakaa osiin. Kuvassa 8 on esitetty esimerkki abstraktista syntaksipuusta.

7.2 Ohjelmointikielen kielioppi ja tulkinta

Xtext-kehyksessä ohjelmointikielen kielioppi määritellään kielioppimäärittelyiden avulla Grammar-kielellä. Kielioppimäärittely sisältää joukon sääntöjä, joita kielentunnistaja vertaa kirjoitettuun ohjelmakoodiin. Säännöt voivat olla leksikaali- (engl. terminal rule) tai jäsentelysääntöjä (engl. parser rule). Jokainen Xtext-kehyksen kielioppimäärittelytiedosto alkaa jäsentelysäännöllä, joka määrittelee koko ohjelmointikielen ja jokaisen sitä edeltävän säännön tulee viitata siihen, jotta ne otettaisiin huomioon ohjelmointikielen

tulkinnassa. Ensimmäinen sääntö toimii myös abstraktin syntaksipuun juurena. Kielentunnistuksen aikana säännöt tulkitaan nelivaiheisen prosessin kautta:

1. Leksikaalinen analyysi (engl. lexing)
2. Jäsentely (engl. parsing)
3. Linkkaus (engl. linking)
4. Validointi (engl. validation)

Ensimmäisen vaiheen aikana joukko merkkijonoja muunnetaan tokeneiksi leksikaalisääntöjen perusteella. Tokeni on merkkijonolle annettu nimi. Toisen vaiheen aikana jäsentelysääntöjen perusteella rakennetaan lähdekoodin abstrakti syntaksipuu. Linkkauksen aikana tulkitaan sääntöjen väliset viitteet. Validoinnin aikana tehdään lisätarkastuksia kielelle, jonka aikana voidaan esimerkiksi tarjota lisäinformaatiota liittyen ohjelmointivirheisiin. Editorin kehittäjä voi itse toteuttaa Java- tai Xtend-ohjelmointikielellä oman validoijan ja näin tarjota monimutkaisempaa syntaksin tarkistusta ohjelmointikielelle. [9]

7.3 ANTLR

Xtext hyödyntää ohjelmointikielen jäsentelijän generoinnissa työkalua nimeltään ANTLR (engl. Another Tool for Language Recognition). Xtext käyttää ANTLR versiota 3. [9] ANTLR 1.0 julkaistiin vuonna 1992 [30] ja sitä kehitetään edelleen San Fransiscon yliopistossa professori Terence Parrin toimesta. Sen kolmas versio on vapaa ohjelmisto, joka on julkaistu BSD-lisenssin alaisena. [22]

ANTLR generoi kielentunnistimen (engl. language recognizer) kielioppimäärittelyn perusteella. Kielentunnistaja on ohjelma, joka tulkitsee kirjoitettua ohjelmointikieltä ja ilmoittaa muun muassa ohjelmointivirheistä. ANTLR-työkalulla generoitu kielentunnistaja kykenee vain oikea rekursiiviseen jäsentelyyn[5] [22]. Tästä johtue myös Xtext-kielioppimäärittely on hyvä suunnitella oikea rekursiivisesti. Xtext-kehyksen backtracktuki mahdollistaa ei-oikea rekursiivisen kielioppimäärittelyn, mutta tehokkuussyistä sen käyttö ei ole suositeltavaa. Vaihtoehtoisesti voidaan hyödyntää Xtext-kehyksen syntaktisia predikaatteja, jotka voivat olla hieman tehokkaampia. Niiden avulla ilmoitetaan mihin asti kielentunnistimen pitää vilkaista ennen päätöksentekoa. [9]

ANTLR hyödyntää omaa kielioppimäärittelytapaansa. Xtext generoi ANTLR-kielioppimäärittelyn sen omasta kielioppimäärittelystä. ANTLR 3 kykenee generoimaan kielentunnistimia seuraaville ohjelmointikielille: Ada95, ActionScript, C, C#, Java, JavaScript, Objective-C, Perl, Python, Ruby ja Standard ML. ANTLR kykenee generoimaan leksikaalisia analysaattoreita, kielenjäsentelijöitä, puujäsentelijöitä sekä leksikaali-jäsentelöitä, joissa leksikaalinen analyysi on yhdistettynä kielijäsentelijän toimintaan.

5 Oikea rekursiivinen jäsentelijä tulkitsee lähdekoodin vasemmalta oikealle.

[22] Kuvassa 7 on esitetty lohkokaavio siitä, miten Xtext-kielioppimäärittely muunnetaan automaattisesti kielentunnistimeksi.

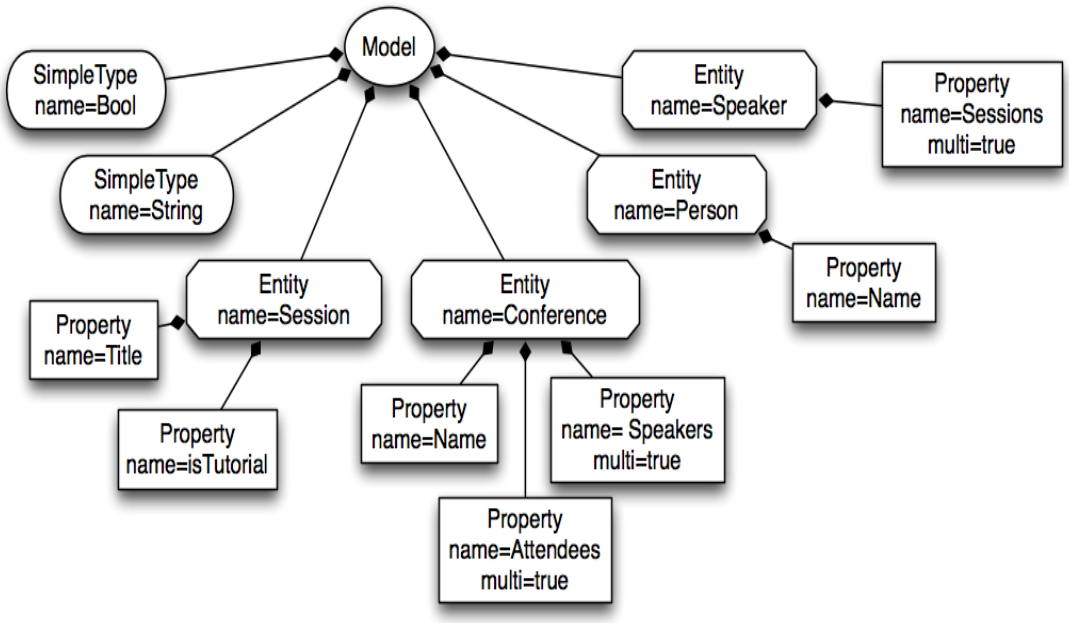
Kuva 7: Xtext-kieliopin muuntaminen kielentunnistimeksi

Ensimmäiseksi Xtext-kielioppi muunnetaan Xtextin toimesta ANTLR-kieliopiksi, josta muodostetaan leksikaalinen analysaattori, jäsentelijä, linkkeri ja validoija, jotka yhdessä muodostavat kielentunnistimen.

7.4 EMF

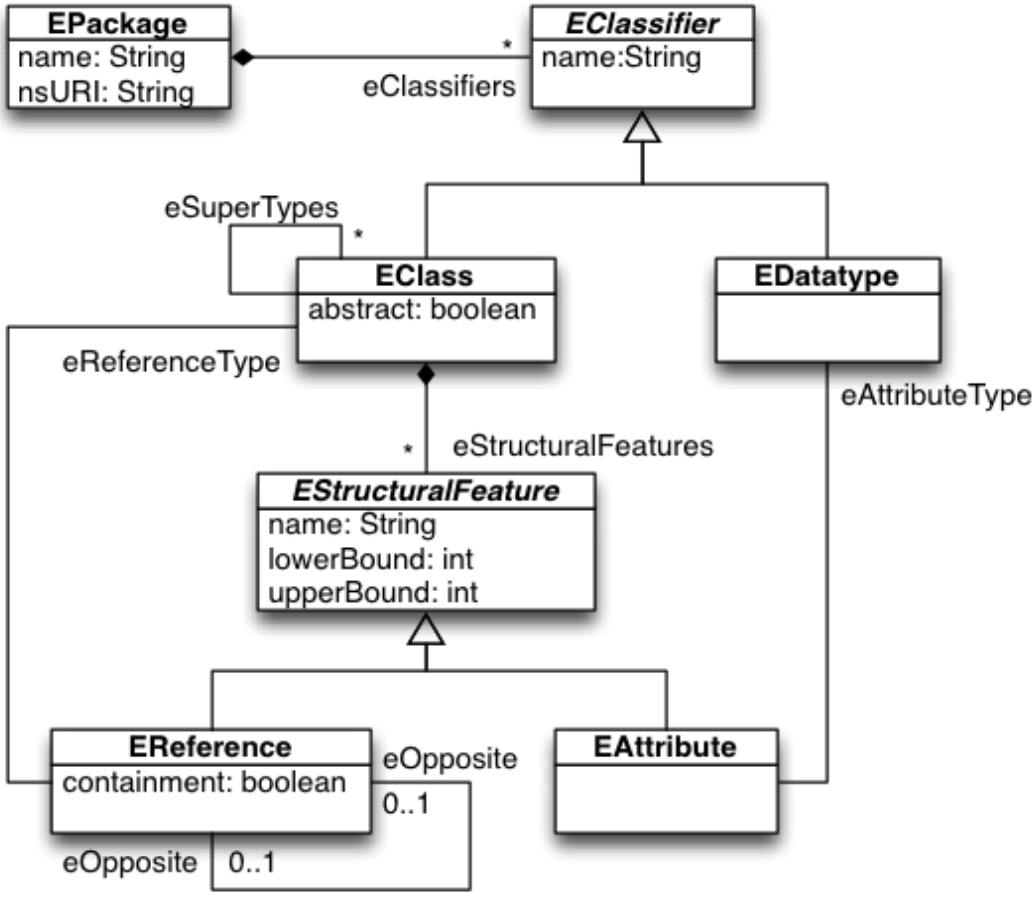
EMF (engl. Eclipse Modeling Framework) on Eclipse-alustan päällä toimiva mallinnuskehys ja lähdekoodin generointityökalu, jota käytetään rakenteelliseen datamalliin perustuvien ohjelmistojen rakentamiseen. EMF-malli on XMI-standardin mukainen. EMF tarjoaa joukon Java- ja adapteriluokkia, jotka mahdollistavat mallin katselun ja komentopohjaisen muokkauksen. Lisäksi se tarjoaa yksinkertaisen editorin. [10]

Xtext hyödyntää EMF-kehystä muodostaakseen ajonaikaisen mallin editorilla kirjoitetusta lähdekoodista. Tätä ajonaikaista mallia kutsutaan tilanteesta riippuen abstraktiksi syntaksipuuksi (engl. abstract syntax tree, lyh. AST) tai dokumentti-objektigraafiksi (engl. document object tree, lyh. DOM), semanttiseksi malliksi tai yksinkertaisesti vain malliksi. [9] Abstraktin syntaksipuun olioita voidaan käsitellä kielen validoinnin aikana. Kuvassa 8 on esitetty esimerkki-AST.



Kuva 8: Esimerkki abstrakti syntaksipuusta [9]

Esimerkissä on Xtext-kehyksen muodostama abstrakti syntaksipuu kirjoitetusta lähdekoodista. Lähdekoodin ohjelmointikieli koostuu *Entity*-määrittelyistä, joilla on *name*-ominaisuus. *Entity* voi sisältää *Property*-määrittelyjä. *Propertyllä* voi olla *name*- ja *multi*-ominaisuuksia. Metamalli määrittelee tavan, jolla editori rakentaa kirjoitetusta tekstistä abstraktin syntaksipuun. Metamalli määritellään ohjelmointikielen kielioppimäärittelyiden avulla hyödyntämällä Xtext-kehyksen Grammar-kieltä. Kielioppimäärittelyiden avulla luodaan ECore-olioista koostuva abstrakti syntaksipuu. [9] Kuvassa 9 on esitetty kaavio Ecoren rakenteesta.



Kuva 9: Ecore [9]

Metamalli määrittelee semanttisten solmujen tyypit EClassin instanssina. EClass voidaan periyttää toisesta EClassista. EClasseilla voi olla EAtrribuutteja ominaisuuksien määrittelemiseksi. EAttribuutin tietotyypin määrittelee EDataType. ECoren mukana tulee joitakin alkeellisia tietotyyppejä, jotka viittaavat Javan tietotyyppeihin. Java-tyyppien erottamiseksi Ecoren tyypeistä, Ecoren tyyppien prefiksinä on ”E”, esimerkiksi EBoolean ja EString. EClassin ominaisuuksia voidaan kuvata EAttribuutilla ja EReferencsillä, joiden molempien kantaluokka on EStructuralFeature. EReference-oliot viittaavat EClass-olioihin. EReference-olion contaiment-lippu kertoo, onko viite contaiment-viite vai ristiviite. EReferenceissä voi edetä vain omistajasta EReference-viitteen viittamaan olioon. Kaksisuuntaiset viitteet määritellään kahdella viitteellä, jossa kumpikin omistaja on toisensa EOpposite. EDataTypen ja EClassin kantaluokka on EClassifier, joka sisältyy EPackageen.

7.5 Xtext-kehyksen ohjelmallinen laajentaminen

Tilanteissa, joissa ohjelmointikieltä ei voida mallintaa täydellisesti Xtext-kehyksen Grammar-kielellä, voidaan hyödyntää ohjelmallista laajennettavuutta. Xtext-kehyksen ohjelmallinen laajentaminen voidaan toteuttaa Xtend- tai Java-kielellä. Xtend kääntyy Java-kieleksi, mistä johtuen niitä voidaan myös käyttää sekaisin. [9] Oletuksena Xtext käyttää Xtend-ohjelmointikieltä.

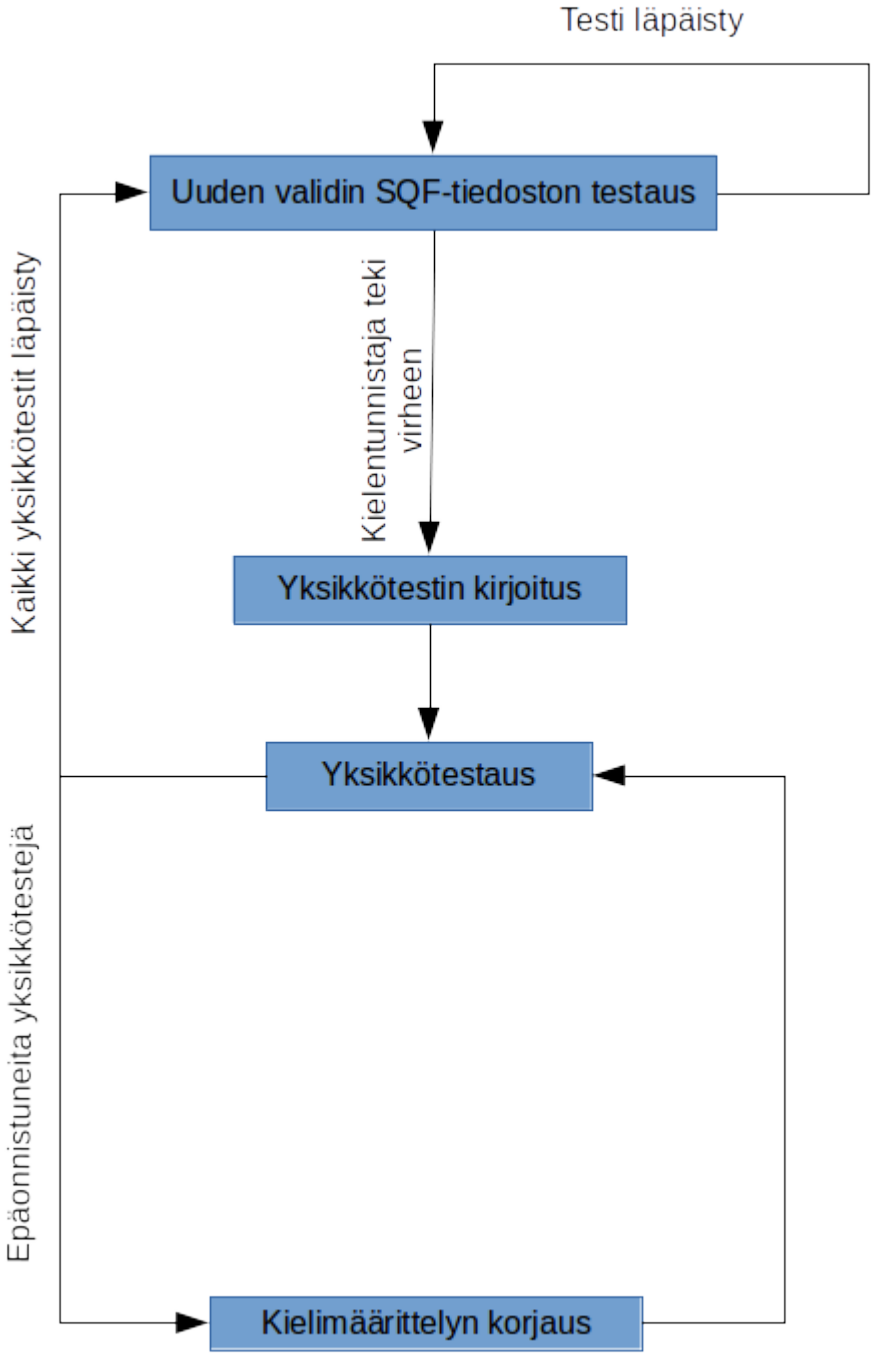
Laajennettavuuden mahdollistamiseksi kaikki Xtext-kehyksen ohjelmistokomponentit hyödyntävät riippuvuusinjektiosuunnittelumallia, mistä johtuen laajennuksen tekijä kykenee muuttamaan riippuvuuden injektoinnin ja täten vaihtamaan koko komponentin esimerkiksi periytettyyn luokkaan. Riippuvuusinjektio on totutettu hyödyntämällä Google Guice -kehystä. Riippuvuuksien ylikirjoittaminen tapahtuu Moduuliluokkaa muokkaamalla. Moduuliluokkia on normaalissa Xtext-projektissa kaksi: yksi Eclipse-alustan käyttöliittymäominaisuuksille ja yksi kielioppimäärittelyille. [9]

8 KEHITYSPROSESSI

Side-kehitysympäristön ohjelmistokehityksessä sovelletaan osa-alueita perinteisestä vesiputousmallista ketteriin menetelmiin. Aluksi määritellään käyttäjävaatimukset lukujen 2-5 tutkimustuloksien perusteella. Tämän jälkeen ohjelma määritellään ja suunnitellaan pintapuolisesti, mutta keskeisimmät suunnittelupäätökset tapahtuivat vasta toteutusvaiheessa. Tärkeän päätöksen lykkääminen viimeiseen sopivaan hetkeen on yksi Lean-periaatteista [13]. Voidaan sanoa, että ohjelman kehitys noudattelee osittain suihkulähdemallia, jonka mukaan ohjelman osittainen suunnittelu voidaan myös tehdä toteutuksen jälkeen [31]. Myöhäiset ohjelmistoarkkitehtuuripäätökset eivät aiheuta ongelmia, koska käytetyt kehykset ja alustat rajoittivat mahdollisia arkkitehtuuripäätöksiä sen tasoisesti, että tarkka suunnittelu ei ole tarpeellista ennen toteutusvaihetta. Side-kehitysympäristön kehityksessä on kaksi vallitsevaa periaatetta: MVP (engl. minimum viable product, luku 8.2) ja testivetoinen kehitys. MVP-ajatusmalli vallitsee koko kehityksen laajuudella, kun taas testivetoinen kehitys keskittyy lähinnä SQF-kielioppimäärittelyiden toteutukseen.

8.1 Testivetoinen kehitys

Side-kehitysympäristön kielentunnistaja perustuu Xtext-kielioppimäärittelyihin. Ensimmäiseksi tehtiin raakaversio kielioppimäärittelyistä, jonka jälkeen sovellettiin testivetoista kehitystä kuvan 10 mukaisesti.



Kuva 10: kielioppimäärittelyiden kehitysprosessi

Kielioppimäärittely pyritään iteroimaan mahdollisimman tarkaksi. Tämä tapahtuu tulkitsemalla yhteisön tekemiä SQF-lähdekooditiedostoja ja testaamalla, että kyseiset tiedostot läpäisevät syntaksin tarkastajan. Tilanteissa, joissa syntaksin tarkastaja ilmoittaa virheistä, mutta SQF-lähdekooditiedosto on syntaktisesti validi, pyritään ongelma eristämään yksikkötestimuotoon. Uuden yksikkötestin kirjoittamisen jälkeen kielioppimäärit-

telyt korjataan niin, että testi tulee läpäistyksi. Kaikkien yksikkötestien läpäisyn jälkeen palataan takaisin ja etsitään uusi SQF-lähdekooditiedosto testausta varten. SQF mahdollistaa muuttujien määrittelyn verkon ylitse ja makrojen avulla, minkä takia globaaleihin muuttuja- ja funktioviitteisiin liittyvät valevirheet sallitaan. Testauksen toteutuksessa hyödynnetään Xtext-kehyksen omaa yksikkötestaustoiminnallisuutta, mutta pääasiassa käytetään Xtext Utils -kirjaston tarjoamaa laajempaa yksikkötestaustukea.

8.2 MVP

Termi MVP on lyhenne sanoista: ”minimum viable product”. Tässä yhteydessä sillä tarkoitetaan MoSCoW-priorisointitavan[6] mukaisesti kehitettyä ohjelmaa, joka toteuttaa kaikki pakolliset vaatimukset. MVP eli ole aikariippumaton, vaan sitä tulisi iteroida koko ohjelmistokehityksen ajan. [13] MVP-periaatteen ideana on tuottaa mahdollisimman pienellä ohjelmistokehitystyöllä mahdollisimman arvokas ohjelma.

Lean-periaatteeseen kuuluu käsitys jätteenpoistosta. Tämän periaatteen mukaan turhan työn minimointi mahdollisimman aikaisessa vaiheessa on tärkeää [13]. Turhan ohjelmointijätteen minimoimiseksi on oleellista, että ohjelman vaatimusmäärittely on mahdollisimman yksityiskohtainen, koska laaja vaatimus voi sisältää matalan ja korkean prioriteetin osavaatimuksia. Matalan prioriteetin vaatimukset voivat olla turhia, joten niiden toteuttamista tulisi välttää.

6 Neljän tason priorisointitapa, josta kerrotaan tarkemmin luvussa 9.

9 KÄYTTÄJÄVAATIMUKSET

Vaatimusmäärittelyllä on yksi tärkeimmistä tekijöistä ohjelmiston toteutuksen onnistumisessa. Ohjelmiston vaatimusmäärittelyn suhteen ketterät menetelmät eroavat perinteisestä vesiputousmallista niin, että ainoastaan vesiputousmallissa pyritään määrittelemään kaikki vaatimukset projektin alussa. Ongelmana vesiputousmallin lähestymistavassa on, että asiakas ei välttämättä tiedä dokumentaation perusteella, miltä ohjelma tulee valmistuttuaan näyttämään. Tästä johtuen vaatimuksien spekulointi ohjelmistokehitystyön alkuvaiheessa voi johtaa turhien tai vääränlaisten ominaisuuksien toteuttamiseen. Ketterän vaatimuksien määrittelytavan mukaan vaatimukset tulisi määrittää juuri silloin kun niitä tarvitaan (engl. just-in-time). Ketterän tavan valinta johtaa yleensä ohjelman parantuneeseen luotettavuuteen ja virheettömyyteen. [12] Toisaalta vesiputousmallin mukaisessa vaatimuksien määrittelytavassa on myös hyötyjä. Sen avulla saavutetaan yleensä parempi kuva kokonaisohjelmasta jo ennen suunnittelutyötä, mikä vastaavasti johtaa parempaan aikataulutukseen ja resurssitarpeiden arviointiin.

Side-kehitysympäristön vaatimusmäärittely toteutetaan hybridinä, jossa yhdistyvät vesiputousmallin ja ketterien menetelmien vaatimusmäärittelytavat. Tämä tarkoittaa, että suurin osa vaatimuksista pyritään esittämään ohjelman toteutuksen alkuvaiheessa. Tavoitteena on sekä vesiputousmallin että ketterien menetelmien hyvien ominaisuuksien yhdistäminen. Perustavanlaatuinen vaatimusmäärittely ohjelman toteutuksen alkuvaiheessa mahdollistaa ohjelmiston perustavalaatuisen suunnittelun jo alkuvaiheessa. Tavoitteena on ohjelman vision tarkka muodostaminen jo toteutustyön alkuvaiheessa. Ketterät menetelmät kuten Scrum nojaavat käsitykseen Product Backlogista, jolla tarkoitetaan listaa vaadituista ominaisuuksista [13]. Perustavalaatuinen alkuvaiheen suunnitteluja määrittely mahdollistavat tarkan Product Backlogin muodostamisen. Side-kehitysympäristön käyttäjävaatimuksien pohjana on käytetty edellä esiteltyjä tutkimustuloksia. Testiympäristön määrittelyt löytyvät taulukosta 5. Jokainen käyttäjävaatimus ja sen kuvaus on esitetty taulukossa 6. Taulukossa prioriteetti 1 on tärkein ja 4 vähiten tärkeä.

Taulukko 5: Testiympäristö

Käyttöjärjestelmä	Windows 7
Prosessori	Intel(R) Core(TM) i5-2500k
Keskusmuisti	8 GB, DDR3, 1600Mhz
Kiintolevy	240 GB SSD

Taulukko 6: Käyttäjävaatimukset

n	Nimi	Kuvaus	Riippuvuus	Prioriteetti
1	Lähdekoodin tulkinta	Kehitysympäristö jäsentelee lähdekoodia ja muodostaa siitä ajonaikaisen muistirakenteen.		1
2	Ohjelmointivirheiden ilmoitus	Ilmoitetaan varmat virheet kuten puolipisteen puuttuminen.	1	1
3	Ilmoitus huonoista tavoista	Ilmoitetaan esimerkiksi selkeistä tehokkuusvirheistä.	1	3
4	Ilmoitus väärin käytetyistä komennoista	Ilmoitetaan, jos komento ei toimi ajonaikaisesti.	1	3
5	Julkisten funktioiden listaus	Kehitysympäristö kokoaa listan funktiosta ja muodostaa siitä näkymän.	1	3
6	Integraatio Arma 3 -pelin kanssa	Esimerkiksi mahdollisuus ajaa lähdekoodia pelissä suoraan kehitysympäristön käyttöliittymän kautta.		2
7	Käyttöliittymäeditori	Editori, jolla voidaan graafisessa ympäristössä muokata käyttöliittymän ulkoasua.		2
8	Uusi tehtävä	Avustaja, joka luo projektin pohjaksi Arma 3 -tehtävän.		3
9	Uusi modi	Avustaja, joka luo projektin pohjaksi Arma 3 -modin.		2
10	Tehtävän muokkaus	Kehitysympäristö voi avata olemassa olevan tehtävän ja siihen liittyvät lähdekoodi- ja asetustiedostot.		1
11	Modin muokkaus	Kehitysympäristö voi avata olemassa olevan modin ja siihen liittyvät lähdekoodi- ja asetustiedostot.	21	2
12	Asetukset	Asetusvalikko		1

n	Nimi	Kuvaus	Riippuvuus	Prioriteetti
13	Lähdekoodin generointi	Toistuvat lähdekoodimuokkaukset voidaan generoida.		3
14	Isojen asetustiedostojen muokattavuus	Side-kehitysympäristön tulee kyetä muokkaamaan 9000-rivin asetustiedostoja ilman, että ohjelma ajautuu käyttökelvottomaan tilaan yli 30 sekunniksi.	1	2
15	Isojen SQF-lähdekooditiedostojen muokattavuus	Kehitysympäristön tulee kyetä muokkaamaan 1000-rivin SQF-lähdekooditiedostoja ilman, että ohjelma ajautuu käyttämättömään tilaan yli 30 sekunniksi.	1	2
16	Tekijän tiedot	Tekijän tiedot tulee saada näkyviin ohjelmassa.		1
17	Verkkotuki	Mahdollisuus muokata lähdekoodia toisella tietokoneella ja ajaa se toisella tietokoneella olevassa pelissä.	6	3
18	Laajennettavuus	Kehitysympäristöä voidaan laajentaa kolmannen osapuolen toimesta.		3
19	Sisältöavustus	Kehitysympäristö näyttää tietoa varatuista sanoista.		1
20	Kirjastotuki	Kehitysympäristössä on mahdollisuus sisällyttää modien funktioita projektiin.	1	3
21	PBO-arkistotuki	Mahdollisuus avata ja muokata PBO-arkistojen sisältöä.		1

Vaatimuksien priorisoinnissa on käytetty MoSCoW-periaatetta, jonka mukaan vaatimukset priorisoidaan neljään kategoriaan. Nämä kategoriat ovat:

1. **Pakko saada (engl. must have)**: Ne vaatimukset jotka on määritelty pakollisiksi. Pakolliset vaatimukset täytyy sisällyttää vaatimuksen määrittelyhetken aikai-

seen aikarajaan. Jos yksikin pakollinen ominaisuus puuttuu valmiista ohjelmasta, pidetään ohjelmaa epäonnistuneena.

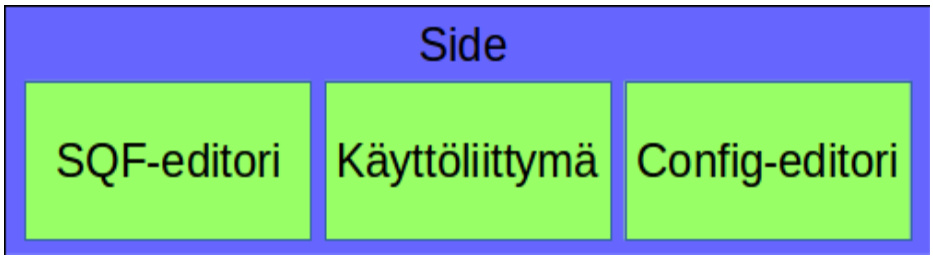
2. **Pitäisi saada (engl. should have)**: Vaatimus pitäisi täyttää, mutta ohjelmistoprojektin onnistuminen ei riipu siitä. Vaatimus on kuitenkin kriittinen ohjelman menestymisen suhteen. Vaatimusta ei ole pakko sisällyttää sen hetkiseen aikatauluun.
3. **Voisi saada (engl. could have)**: Voisi-ominaisuudet ovat vähemmän kriittisiä ja usein niitä pidetään asioina, jotka olisi mukava olla.
4. **Ei saada (engl. would not have)**: Alimman prioriteetin ominaisuuksia, joiden takaisinmaksukyky on huonoin, eikä niitä tulisi sijoittaa ohjelmaan missään sen vaiheessa. [12]

Ohjelmistokehitystyön hyödyn maksimoimiseksi vaatimukset tulisi toteuttaa niin, että korkein prioriteetti tehdään ensimmäiseksi, jos sen riippuvuus on täytetty. Aikataulun tiukentuessa voidaan matalan prioriteetin vaatimuksia jättää toteuttamatta.

10 ARKKITEHTUURI

Toteutettava ohjelma on Eclipse-alustan päälle rakennettu integroitu kehitysympäristö SQF-ohjelmointikielelle, joka on erityisesti tarkoitettu Arma 3 -pelin laajentamiseen. Johtuen SQF-ohjelmointikielen nopeasta kehittymisestä ja sen tukemien pelien monista SQF-variaatioista, pyritään SQF-kielioppimäärittelyjen laajennettavuus pitämään mahdollisimman hyvänä. Side-kehitysympäristöä tullaan levittämään stand-alone tyyppisenä sovelluksena. Alustavasti ohjelma suunnitellaan toimimaan Windows-alustalla, mutta Java-ohjelmointikielen ja Eclipse-alustan alustariippumattomuus mahdollistaa ohjelman sovittamisen myös muille käyttöjärjestelmille.

Kuvassa 11 on esitetty Side-kehitysympäristön rakennetta kuvaava kaavio.



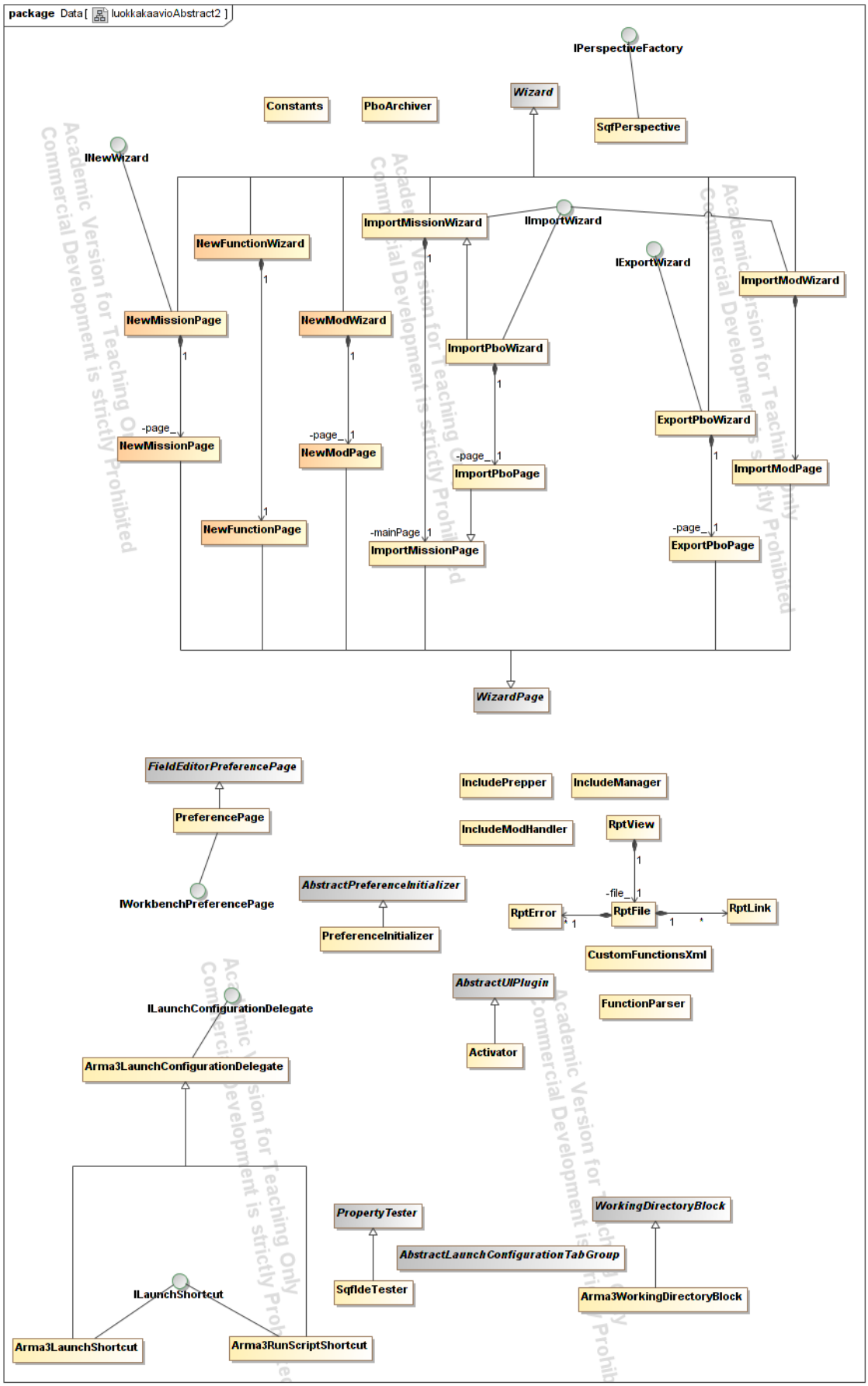
Kuva 11: Side-kehitysympäristön rakenne

Ohjelma koostuu kolmesta Eclipse-ominaisuudesta: SQF-editorista, Config-editorista ja käyttöliittymästä. Editoreilla tarkoitetaan tekstikenttää, johon tekstiä kirjoitetaan, ja siinä näkyviä graafisia olioita. Käyttöliittymällä tarkoitetaan ominaisuuksia kuten avustajia, asetusnäkymiä ja työkalupalkkia. Editorit toteutetaan Xtext-kehystä hyödyntäen ja käyttöliittymä toteutetaan tavanomaista Eclipse-alustaa hyödyntäen. SQF-editori osuus toteutetaan *org.arma.xtext.sqf.feature*-ominaisuudessa (engl. feature), Config-editori toteutetaan *org.arma.xtext.config.feature*-ominaisuudessa ja käyttöliittymä toteutetaan *org.arma.side.ui.feature*-ominaisuudessa. Yhdessä nämä ominaisuudet muodostavat Side nimisen Eclipse-tuotteen (engl. product).

10.1 Käyttöliittymän oliosuunnittelu

Editoreiden lisäksi Side koostuu integroiduille kehitysympäristöille tyypillisestä käyttöliittymätoiminnallisuudesta. Toiminnallisuus on toteutettu *org.arma.side.ui.feature*-ominaisuuden alle. Tyypillisiä käyttöliittymäkomponentin toiminnallisuuksia ovat avustajat (engl. wizards), dialogit ja asetusnäkymät. Lähdekoodin pituuden minimoimiseksi suurin osa Side-kehitysympäristön toiminnallisuudesta on pyritty toteuttamaan laajentamal-

la Eclipse-alustan valmiita osia. Kuvassa 12 on esitetty Side-kehitysympäristön käyttöliittymän luokkakaavio, johon on merkitty harmaalla Eclipse-alustan periytetyt luokat ja käytetyt rajapinnat.



Kuva 12: Luokkakaavio Side-kehitysympäristön käyttöliittymästä

Osa luokkakaavion luokista voidaan jättää edellä esitetyn vaatimuksien priorisoinnin mukaisesti toteuttamatta. Luokkakaavion yläosassa on merkitty ohjelmiston avustajat. Avustajat koostuvat sivuista. Side-kehitysympäristössä avustajina toimivat *NewMissionWizard*, *NewModWizard*, *NewFunctionWizard*, *ImportMissionWizard*, *ImportPboWizard*, *ExportPboWizard* ja *ImportModWizard*. *NewMissionWizardin* luo uuden tehtävän; *NewModWizard* luo uuden modin; *NewFunctionWizard* luo uuden SQF-funktion ja esittelee sen; *ImportMissionWizard* avaa olemassa olevan tehtävän; *ImportPboWizard* avaa PBO-arkistossa olevan tehtävän tai modin; *ExportPboWizard* luo tehtävästä tai modista PBO-arkiston ja *ImportModWizard* avaa olemassa olevan modin.

Luokkakaavion keskiosassa olevat *PreferencePage* ja *PreferenceInitializer* toteuttavat Side-kehitysympäristön asetukset. Vasemmalla keskellä oleva *Arma3LaunchConfigurationDelegate* käsittelee ajamiseen liittyviä toimintoja. Siitä on periytetty *Arma3RunScriptShortcut* ja *Arma3LaunchShortcut* luokat. *Arma3RunScriptShortcut* mahdollistaa SQF-lähdekooditiedoston ajamisen pelin ollessa päällä. *Arma3LaunchShortcut* toteuttaa kehitettävän modin tai tehtävän käynnistämisen pelin kanssa. Lisäksi se käynnistää pelin dynaamisen kirjaston kanssa, mikä mahdollistaa *Arma3ScriptShortcut*-luokan toiminnallisuuden. Lokitiedostojen käsittelyyn Side-kehitysympäristöstä löytyy kaavion oikeasta alalaidasta luokat *RptFile, RptError* ja *RptLink*. *RptFile* mallintaa Arma 3 -lokitiedostoa. Se voi sisältää satunnaisen määrän virheilmoituksia, joita kuvataan *RptError*-luokalla. *RptError* pitää kirjaa virheilmoituksista. *RptLink* mallintaa lokitiedoston viitteitä SQF-lähdekooditiedostoihin. *Constants* sisältää globaalit vakiot. *PboArchiver* mahdollistaa PBO-arkistojen käsittelyn. Kaavion vasemmasta alakulmasta löytyvät *FunctionParser* ja *CustomFunctionXml* mahdollistavat SQF-funktioiden tunnistuksen. *Activator* ajaa kehitysympäristön käynnistyksen aikaiset toiminnot. *SqfIdeTester* varmistaa, että tietyt kontekstivalikkovaihtoehdot näytetään vain tietyille elementeille. *IncludeModHandler* ja *IncludePrepper* toteuttaa modin lähdekoodin sisäyllytyksen kirjastotyyppisesti. *SqfPerspective* toteuttaa Side-kehitysympäristön näkymän personalisoinnin. Osa edellä mainituista luokista laajentaa Eclipse-alustaa laajennuspisteiden kautta. Laajennuspisteistä kerrotaan tarkemmin seuraavassa alaluvussa.

10.2 Laajennussuunnittelu

Side on Eclipse-alustan päälle kehitettävä lisäosa. Laajennussuunnittelulla tarkoitetaan tässä yhteydessä Eclipse-alustan laajentamisen suunnittelua. Eclipse-alustaa laajennetaan laajennuspisteiden kautta, jotka määritellään *plugins.xml*-tiedostoissa. Side-kehitysympäristön käyttämät laajennuspisteet on esitetty taulukossa 7.

Taulukko 7: laajennuspisteet

Laajennuspisteet	Tarkoitus	Vaatimukset
org.eclipse.core.expressions.definitions	Editorit (Xtext)	1, 2, 3, 4, 20, 21
org.eclipse.core.resources.markers	Editorit (Xtext)	1, 2, 3, 4, 19, 20
org.eclipse.core.runtime.preferences	Asetukset	11, 12
org.eclipse.core.runtime.products	Tuotteen nimi	17
org.eclipse.debug.ui.launchConfiguration TabGroups	Pelin käynnistys	6, 8, 9
org.eclipse.debug.ui.launchConfiguration TypeImages	Pelin käynnistys	6, 8, 9
org.eclipse.debug.ui.launchShortcuts	Pelin käynnistys ja lähdekooditiedoston ajaminen	6, 8, 9, 17
org.eclipse.help.toc	Editorit (Xtext)	1, 2, 3, 4
org.eclipse.ui.commands	Tehtävän avaaminen ja funktiokirjastotuki	20
org.eclipse.ui.editors	Editorit (Xtext)	1, 2, 3, 4, 19, 20
org.eclipse.ui.editors.documentProviders	Editorit (Xtext)	1, 2, 3, 4, 19, 20
org.eclipse.ui.editors.templates	Asetukset	12
org.eclipse.ui.exportWizards	PBO-arkistojen hallinta	21
org.eclipse.ui.handlers	Editorit (Xtext)	1, 2, 3, 4, 19
org.eclipse.ui.ide.markerResolutions	Editorit (Xtext)	1, 2, 3, 4, 19
org.eclipse.ui.importWizards	Tehtävän avaaminen	
org.eclipse.ui.keywords	Editorit (Xtext)	1, 2, 3, 4, 19, 20
org.eclipse.ui.menus	Työkalupalkin pikanäppäimet	10, 11
org.eclipse.ui.newWizards	Uusi tehtävä tai modi avustajat	8, 9
org.eclipse.ui.perspectiveExtensions	Personalisoitu ulkoasu	5, 6
org.eclipse.ui.perspectives	Personalisoitu ulkoasu	5, 6
org.eclipse.ui.preferencePages	Asetukset	11
org.eclipse.ui.propertyPages	Funktiokirjastotuki	20
org.eclipse.ui.views	Funktionäkymä	5

Taulukon ensimmäisessä sarakkeessa on esitetty laajennuspisteen tunnus; toisessa sarakkeessa kerrotaan, miten laajennuspistettä käytetään ja kolmannessa sarakkeessa on esitetty vaatimus (luku 9, taulukko 6), joka täyttyy laajennuksen ansiosta. Laajennuspisteet vaikuttavat ohjelman luokkarakenteeseen, niin että osalle laajennuspisteelle täytyy kehittää laajennuksen tekevä luokka Side-kehitysympäristössä. Laajentavan luokan tulee myös toteuttaa tietty Eclipse-alustan rajapinta, jonka kautta Eclipse kutsuu lisäosaa. Laajennukset, joiden tarkoitus on ”Editorit (Xtext)” ovat Xtext-kielioppimäärittelyistä generoituja laajennuksia, joihin ei oteta yksityiskohtaisesti kantaa kapselointiperiaatteen mukaisesti. Taulukosta on jätetty joitakin alemman prioriteetin vaatimuksia pois. Laadullisiin vaatimuksiin ei ole otettu kantaa.

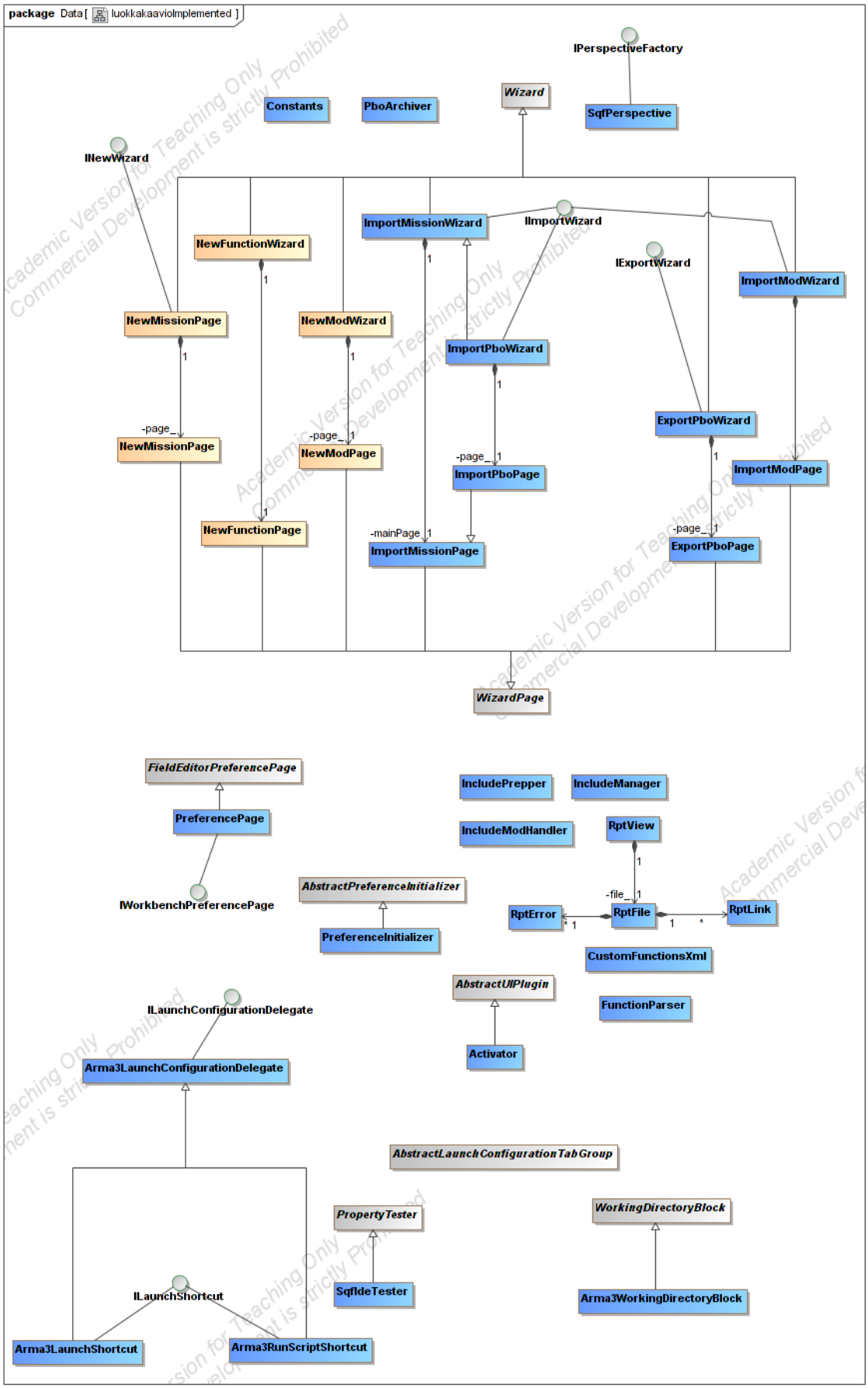
11 TOTEUTUS

Side noudattaa tyypillisten Eclipse-alustaan perustuvien kehitysympäristöjen ratkaisuja, jolloin Eclipse itsessään määrittää ohjelman toiminnallisuuden suurelta osin, mistä johtuen ohjelmaa ei ole pyritty suunnittelemaan yhtä yksityiskohtaisesti kuin tavanomaista ohjelmistoa. Tässä kappaleessa otetaan pääasiassa kantaa käytettyihin Eclipse-alustan laajennustapoihin ja asioihin, joihin Eclipse ei itse ota kantaa.

Side-kehitysympäristön kaikkia vaatimuksia ei ole toteutettu, vaan osa on karsittu pois priorisointitavan mukaisesti. Taulukossa 8 on esitetty toteuttamatta jääneet vaatimukset ja niiden prioriteetit. Kuvassa 13 on esitetty toteutetut luokat sinisellä, toteuttamatta jääneet keltaisella sekä Eclipse-alustan luokat ja rajapinnat harmaalla.

Taulukko 8: Toteuttamattomat vaatimukset

n	Nimi	Prioriteetti
3	Ilmoitus huonoista tavoista	3
4	Ilmoitus väärin käytetyistä komennoista	3
7	Käyttöliittymäeditori	2
8	Uusi tehtävä	3
9	Uusi modi	2
11	Modin muokkaus	2
13	Lähdekoodin generointi	3



Kuva 13: Toteutuksen luokkakaavio

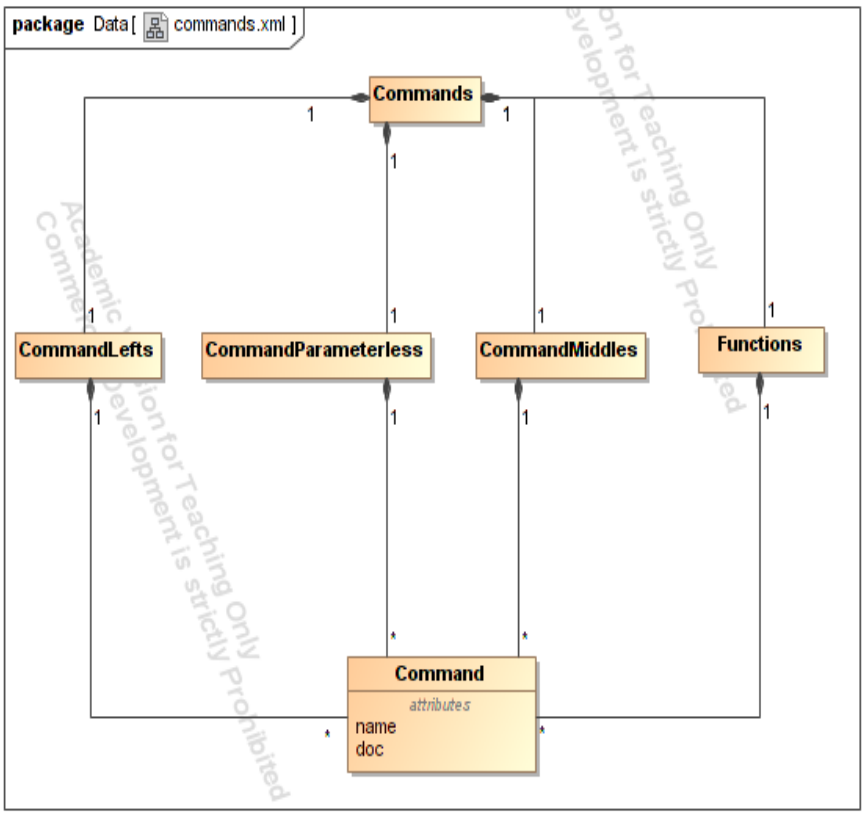
Toteutuksen karsinnasta huolimatta Side on pyritty toteuttamaan kokonaisena käyttökelpoisena ohjelmistona.

Luku koostuu yhdeksästä alaluvusta. Ensimmäinen alaluku kertoo, miten Xtext-kehystä on hyödynnetty editoreiden toteutuksessa; toinen alaluuluku, miten ohjelmointikielen dokumentaatio rakennetaan automaattisesti Bohemia Interactive Community Wiki -sivuston tietojen avulla; kolmas ja neljäs alaluku, miten Side käsittelee PBO-arkistoja; viides alaluku, miten Side-kehitysympäristöön on integroituna SQF-ohjelmointikielen dokumentaatio; kuudes alaluku, miten Side täydentää SQF-funktiot; seitsemäs luku, miten Side kommunikoi ajonaikaisesti pelin kanssa; kahdeksas alaluku Side-kehitysympäristön käyttöliittymästä niiltä osin, kun sitä ei käsitelty ja yhdeksännessä luvussa esitetään päätelmät.

11.1 Editori ja kielen mallinnus

Editorilla tarkoitetaan tässä yhteydessä Side-kehitysympäristön tekstikenttää, johon lähdekoodia kirjoitetaan. Side-kehitysympäristön editori toteutetaan Eclipse-alustan päälle käyttäen apuna Xtext-kehystä. Sen avulla SQF-ohjelmointikielelle ja sen asetustiedostoille määritellään kielioppi, jota hyödynnetään syntaksin tarkastuksessa. Kieliopin määrittelyssä ei ole pyritty täydellisyyteen, vaan sen tarkoituksena on osoittaa yleiset ohjelmointivirheet ja olla ennen kaikkea näyttämättä valevirheitä. Tilanteissa, joissa Xtext ei kykene kieliopin avulla toteuttamaan haluttuja toiminnallisuuksia, sovelletaan Javalla toteutettuja laajennuksia Xtext-syntaksitarkastajaan. Laajennuksista ja kielioppimäärittelyistä saadaan generoitua editori, joka tukee muun muassa syntaksin tarkistusta, automaattista täydennystä ja tooltippejä.

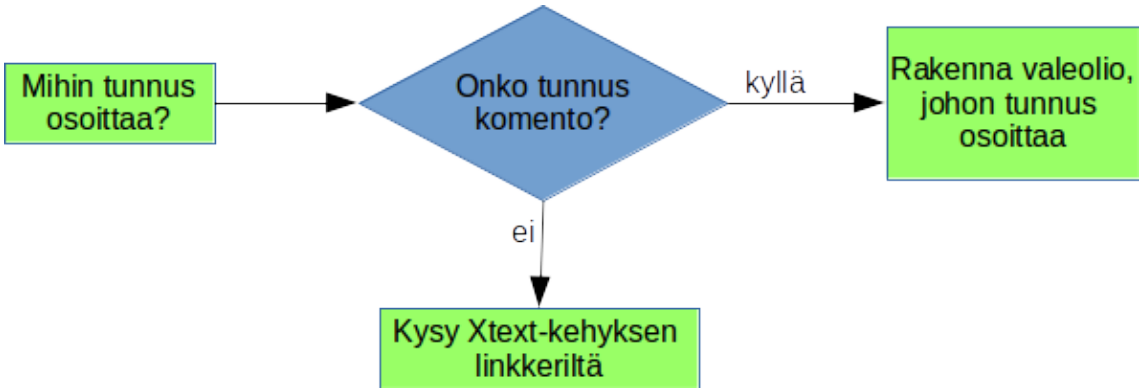
SQF-ohjelmointikieli koostuu lähes 2000 komennosta. Xtext-kehyksen ja Java-ohjelmointikielen rajoituksista johtuen (staattisen alustajan maksimikoko) kaikkia komentoja ei voida sisällyttää suoraan kielioppimäärittelyyn [28]. Sen sijaan komennot listataan erillisessä *commands.xml*-tiedostossa, joka luetaan ohjelman käynnistyessä. Siinä listattujen komentojen avulla tehdään validoinnin aikainen syntaksin tarkistus. Komentojen eriyttäminen kielioppimäärittelyistä mahdollistaa helpon laajennettavuuden kolmannen osapuolen toimesta. Kuvassa 14 on esitetty commands.xml-dokumentin rakenne.



Kuva 14: commands.xml-dokumentin rakenne

commands.xml-dokumentti koostuu komento- ja funktiomäärittelyistä. SQF-komennot ovat jaoteltu vaadittujen parametrien perusteella: CommandLefts-tyyppiset komennot ottavat parametrit komennon jälkeen, jolloin komento on vasemmalla (engl. left) parametreista; CommandParameterless-tyyppiset komennot eivät tue parametreja; CommandMiddles-tyyppiset komennot ottavat parametrin sekä komentoa ennen että sen jälkeen, jolloin komento on parametrien keskellä (engl. middle). Komennolla on name-attribuutti, joka sisältää komennon nimen ja doc-attribuutti, joka sisältää HTML-tyyppisen dokumentaation komennon käytöstä. Functions-rakenne sisältää Arma 3 -pelin sisäiset SQF-funktiot, joilla on samaan tapaan nimi ja dokumentaatio.

SQF-ohjelmointikieli asettaa monia haasteita kielioppimäärittelyille. Yksi näistä haasteista on globaalin muuttujaviitteen syntaktinen identtisyys komentojen kanssa. Xtext-kehyksen syntaksin tarkastaja pyrkii ilmoittamaan muuttujaviitteistä, joille ei löydy määrittelyä. Tätä operaatiota kutsutaan linkkaukseksi. Linkkaus ja validointi ovat erillisiä toimenpiteitä, jolloin validaattori ei voi puuttua linkkausvirheisiin. Side-kehitysympäristössä Xtext-linkkeri on ylikirjoitettu, jotta linkkaus ja validointi voidaan suorittaa samalle tunnukselle. Kuvassa 15 on esitetty vuokaavio uuden linkkerin toiminnasta.



Kuva 15: Side-kehitysympäristön linkkerin toiminta

Uusi linkkeri toteuttaa luokan *SqfLazyLinkingService*, joka kiertää linkkauksen, jos viite on komento. Tällöin tunnus asetetaan osoittamaan valeolioon ja virheilmoitusta ei ilmene editorissa. Xtext antaa oletuksena tooltipit viitteen kohteelle eikä viitteelle, mistä johtuen myös hover-toiminnon toiminnallisuus on ylikirjoitettu uudella luokalla nimeltään *SqfObjectHoverProvider*. Idea on samantyyppinen uuden linkkerin kanssa: jos viite on komento, niin tooltip asetetaan viitteelle eikä viitatulle.

SQF-kielioppimäärittelyiden lisäksi myös Arma 3 -pelin asetustiedostojen syntaksi on mallinnettu, jolloin SQF-lähdekoodista voidaan tehdä ristiviittauksia asetustiedostoissa määriteltyihin luokkiin ja vakioihin. Side kykenee myös ilmoittamaan asetustiedostoissa olevista syntaksivirheistä. Toisin kuin SQF-syntaksin tarkastaja, asetustiedoston syntaksin tarkastaja on toteutettu hyödyntäen pelkästään kielioppimäärittelyjä.

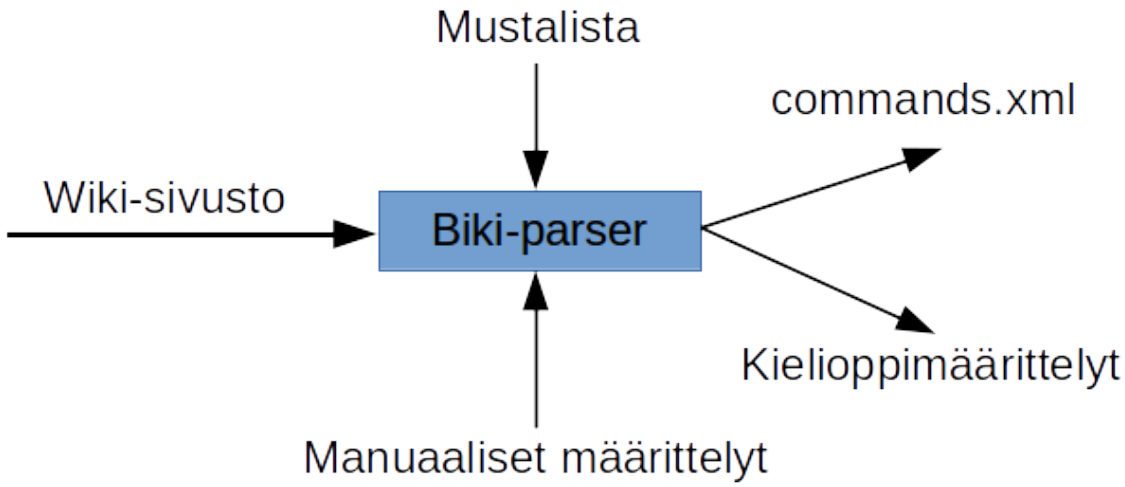
Side ei tarkista käytettävyyden takia funktioviitteitä. Tämä tarkoittaa sitä, että lähdekoodissa ”_fnc_”-tekstin sisältämät tunnukset tunnistetaan aina valideiksi funktiokutsuksi. Side hyödyntää kuitenkin tietoa olemassa olevista globaaleista funktioista automaattisen täydennyksen yhteydessä (luku 11.6). SQF-lähdekoodissa globaaleita funktioita tai muuttujia voidaan määritellä modissa ja tehtävässä. Modin määrittelemä globaali funktio tai muuttuja näkyy käynnissä olevalle tehtävälle ja muille modeille. Side tulkitsee avatun projektin funktiot jäsentämällä asetustiedostojen funktiomäärittelyt ja rakentamalla niistä projektikohtaisen tietorakenteen, jota hyödynnetään automaattisen täydennyksen yhteydessä.

Side tukee viitteitä modien lähdekoodiin. Tämä tarkoittaa sitä, että modissa määritelty funktio tai muuttuja voi olla viitteen kohde kehitettävässä projektissa. Modien funktiot ehdotetaan automaattisen täydennyksen yhteydessä. Modin liittäminen osaksi projektia tapahtuu painamalla hiiren oikealla projektia ja valitsemalla ”Include Functions from Mod”. Liittämisen jälkeen modissa oleva lähdekoodi puretaan projektin includes-kansioon, josta syntaksin tarkastaja löytää sen. Modissa määritellyt globaalit funktiot joudutaan jäsentelemään ohjelmallisesti, jolloin niistä kootaan erillinen projektikohtainen XML-tiedosto, joka on rakenteeltaan samanlainen kuin edellä mainittu *com-*

mands.xml-tiedosto. Tämä tiedosto luetaan automaattisen täydennyksen yhteydessä. Includes-kansiota ei sisällytetä tehtävästä luotuun PBO-arkistoon.

11.2 Biki-parser

SQF-ohjelmointikieli koostuu lähes kahdesta tuhannesta komennosta. Näiden kaikkien komentojen lisääminen yksi kerrallaan kielioppimäärittelyyn olisi työläs tehtävä. Sen sijaan, että kielioppimäärittely tehtäisiin kokonaan manuaalisesti kehitettiin Biki-parser-ohjelmisto, joka jäsentelee Bohemia Interactive Community Wiki -sivustolta löytyvät komentomäärittelyt. Kuvassa 16 on esitetty lohkokaavio Biki-parserin toiminnasta.



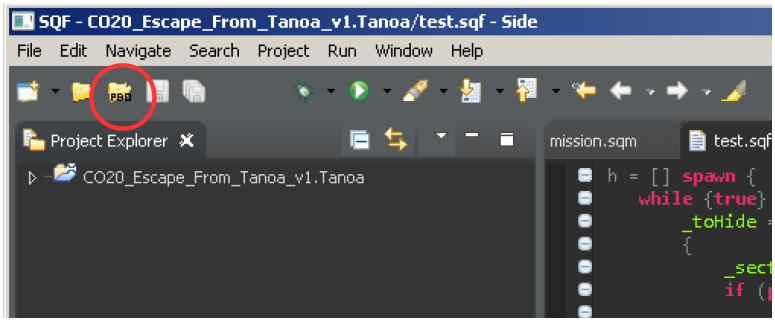
Kuva 16: Biki-parserin toiminta

Pipeline-tyypinen ohjelma ottaa sisäänsä Bohemia Interactive Community Wiki -sivuston paikallisen kopion ja kirjoittaa *commands.xml*-tiedoston ulostulona. *commands.xml*-tiedosto sisältää komentokohtaisen syntaktisen informaation ja sen lisäksi kuvauksen komennoista tooltip-toimintoa varten. Tilanteissa joissa Biki-parser ei onnistu selvittämään komentoa tai sen ominaisuuksia, voidaan komentokohtainen wiki-sivu mustalistata ja lisätä komento manuaalisesti. Tehokkuussyistä johtuen parametrittomat komennot joudutaan myös listaamaan Grammar-tiedostossa, koska sen tyyppinen komento on syntaktisesti identtinen ristiviitettä vaativan tunnuksen kanssa. *commands.xml*-rakenne on helposti ymmärrettävissä ja mahdollistaa täten kielioppimäärittelyiden helpon laajentamisen kolmannen osapuolen toimesta. Kuvassa 14 on esitetty commands.xml-dokumentin rakenne.

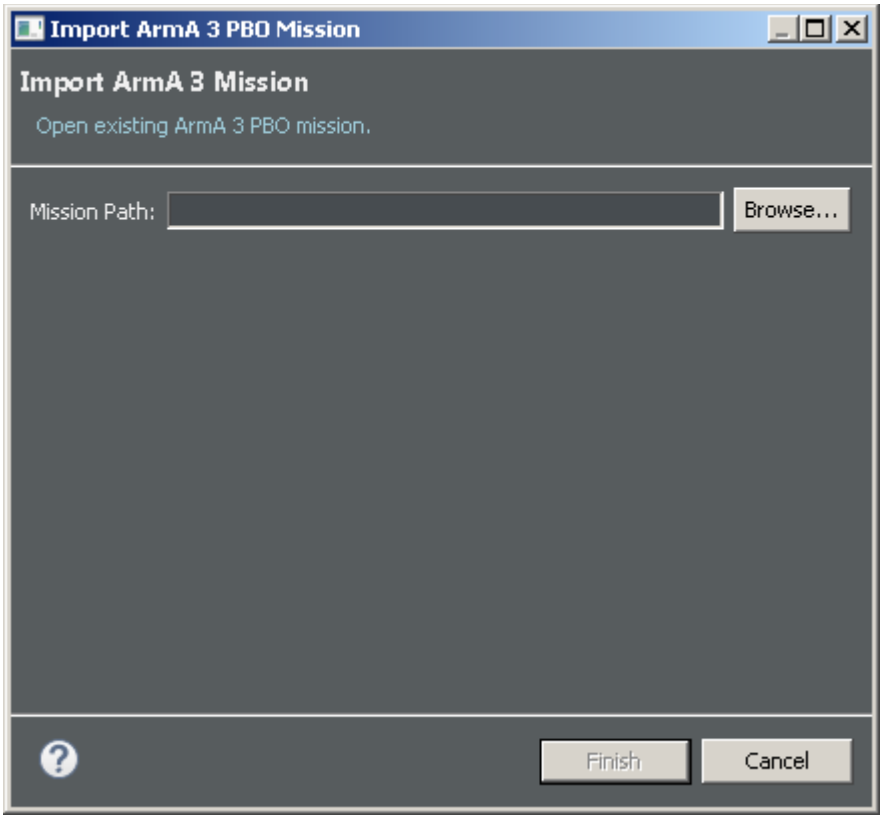
11.3 PBO-arkistojen avaaminen

Arma 3 -tehtävät ja moditiedostot voidaan pakata PBO-arkistoihin. Side-kehitysympäristöön on toteutettu tuki PBO-pakattujen Arma 3 -tehtävien avaamiseen. PBO-arkiston

avaaminen tapahtuu Eclipse-alustan tuontiominaisuuden (engl. import) avulla. PBO-arkiston avaaminen voidaan tehdä joko tavanomaisesti valitsemalla import *file*-valikosta tai painamalla työkalupalkin avaa-kuvaketta, joka on esitetty kuvassa 17. Painamalla kuvaketta avautuu kuvan 18 mukainen näkymä.



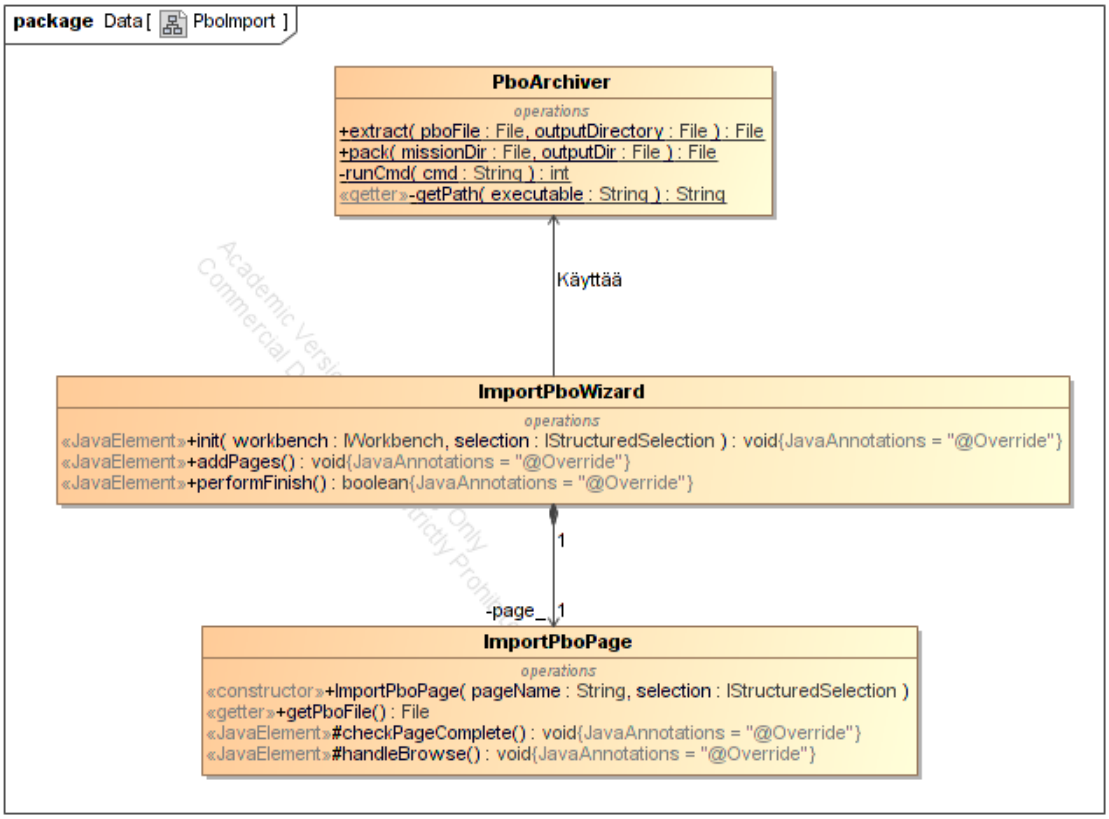
Kuva 17: PBO-kuvake



Kuva 18: PBO-arkiston avaamisen näkymä

Painamalla näkymän *browse*-näppäintä avautuu tiedostonäkymä, jossa käyttäjä voi asettaa polun PBO-arkistolle. *Finish*-näppäin aktivoituu, kun asetettu polku osoittaa avattavaan PBO-arkistoon. Ohjelma purkaa PBO-arkiston sisällön työtilaan, jolloin sen muokkaaminen on mahdollista.

PBO-arkistojen hallinta on toteutettu laajentamalla *org.eclipse.ui.ImportWizards*-laajennuspistettä, jonka toteuttaa *ExportPboWizard*-luokka. Kuvassa 19 on esitetty luokat, jotka liittyvät PBO-arkiston avaamiseen.

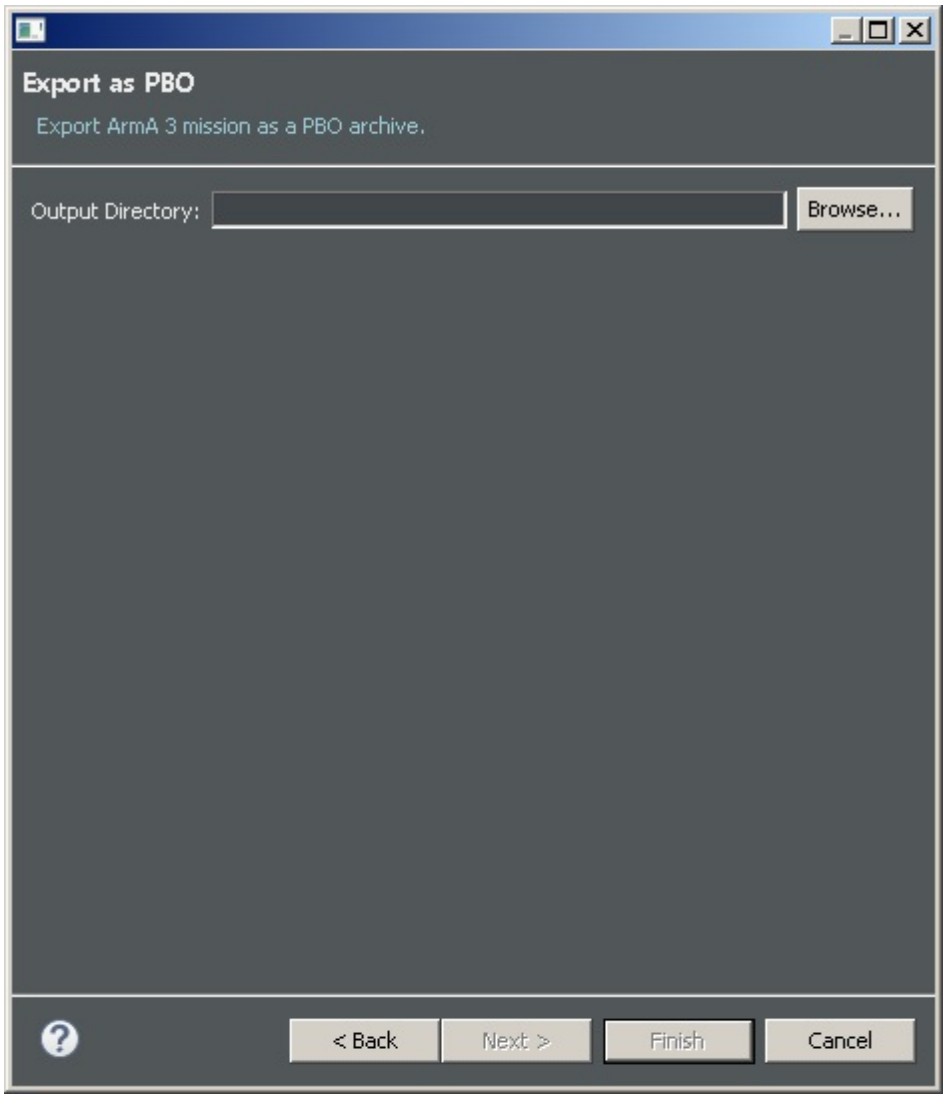


Kuva 19: PBO-arkiston avaamisen luokkakaavio

ImportPboWizard toteuttaa avustajan, jonka avulla PBO-arkisto avataan. Se käyttää hyväkseen *PboArchiver*-luokkaa, joka kutsuu *ExtractPboDos*-komentolinjasovellusta, jonka avulla PBO-arkisto puretaan. *ImportPboPage* toteuttaa näkymän avustajan ainoalle sivulle.

11.4 PBO-arkistojen luonti

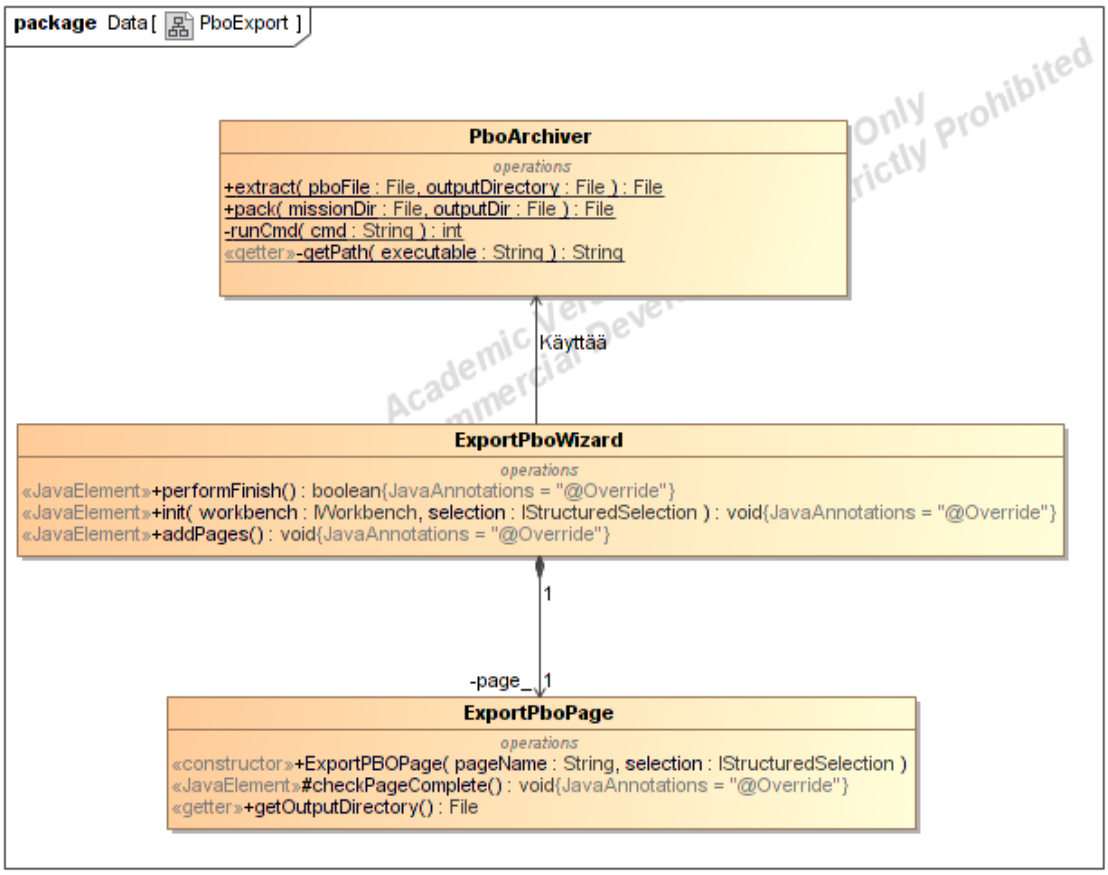
Arma 3 -tehtävät ja moditiedostot voidaan pakata PBO-arkistoihin. Side-kehitysympäristöön on toteutettu tuki Arma 3 -tehtävien PBO-pakkaamiseen. PBO-arkiston luonti tapahtuu Eclipse-alustan vientiominaisuuden (engl. export) kautta. Käyttäjä voi viedä kehitettävän tehtävän PBO-arkistona painamalla tehtävää hiiren oikealla, valitsemalla ”export” ja valitsemalla avautuneesta valikosta ”PBO Archive”. Kuvassa 20 on esitetty näkymä, joka avautuu valinnan jälkeen.



Kuva 20: PBO-arkiston luonnin näkymä

Painamalla *Browse* käyttäjä voi valita kansion, johon PBO-arkisto luodaan. Kansion ollessa validi aktivoituu *Finish*-näppäin, jota painamalla PBO-arkisto luodaan. PBO-arkiston nimeksi asettuu projektin nimi, jolla on päätteenä *pbo*.

PBO-arkistotuki on toteutettu laajentamalla Eclipse-alustan laajennuspistettä *org.eclipse.ui.exportWizards*, jonka toteuttaa luokka *ExportPBOWizard.* Kuvassa 21 on esitetty luokkakaavio PBO-arkistojen hallinasta.

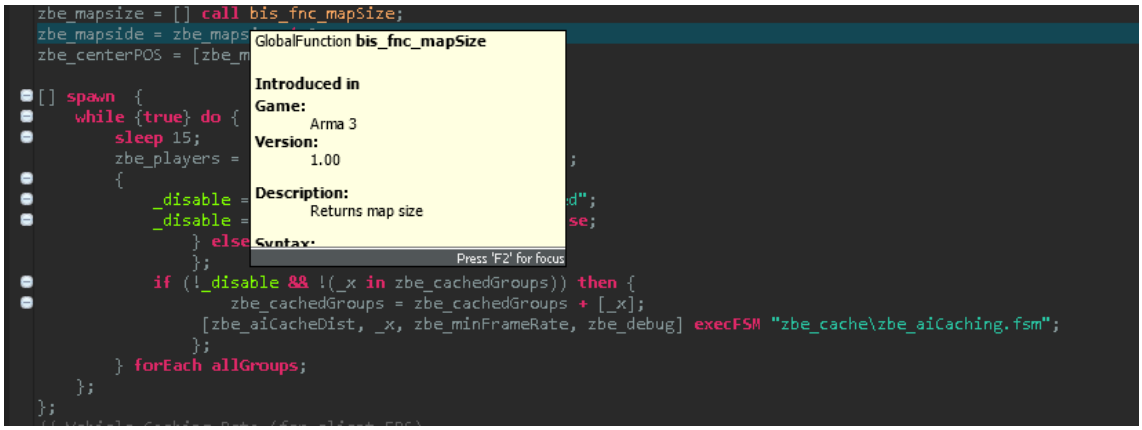


Kuva 21: PBO-arkistojen luonnin luokkaavio

PboArchiver-luokka toteuttaa PBO-arkistojen hallinnan. Se kutsuu kolmannen osapuolen *cpbo*-komentolinjaohjelmaa ja pakkaa sen avulla kansion sisällön PBO-arkistoon. *ExportPboWizard* on avustaja (engl. wizard), jonka avulla Arma 3 -tehtävästä luodaan PBO-arkisto. *ExportPboPage* määrittää näkymän avustajan ainoalle sivulle.

11.5 Ohjelmointikielen dokumentaation integraatio

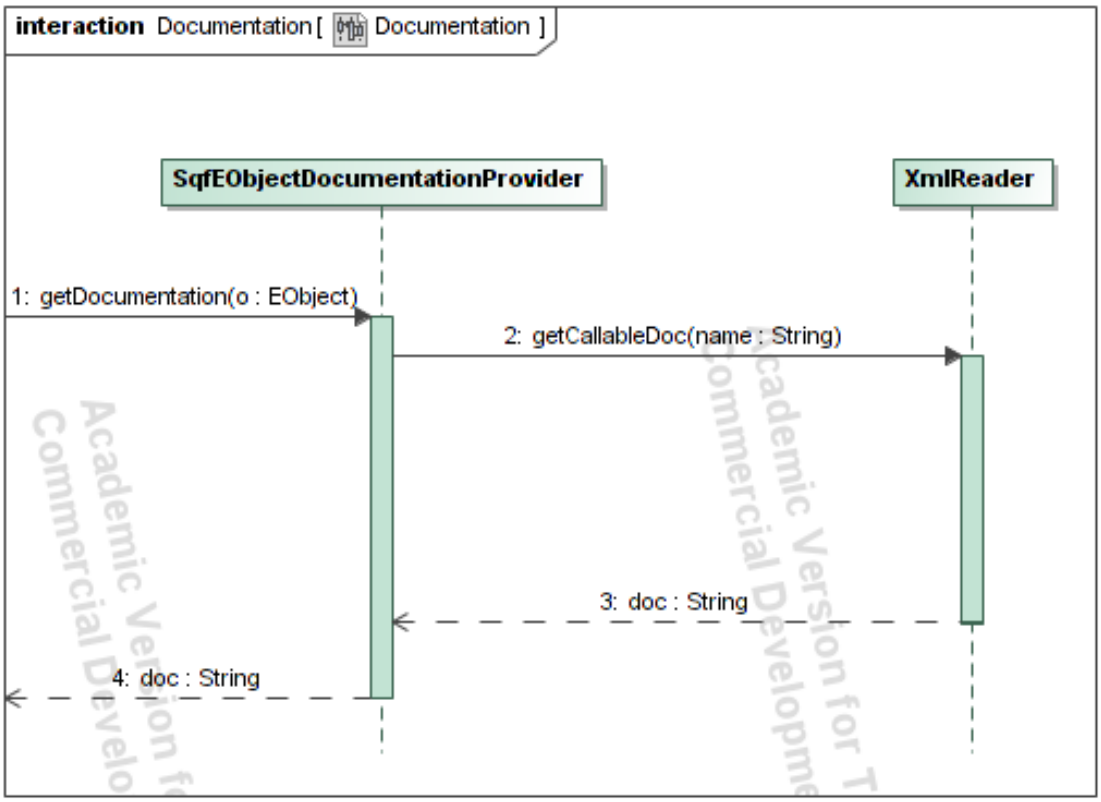
Ohjelmointikielen dokumentaation integraatiolla tarkoitetaan integroidun kehitysympäristön ominaisuutta, joka mahdollistaa ohjelmointikielen komentojen ja funktioiden dokumentaation tarkastelun integroidusta kehitysympäristöstä käsin. Side-kehitysympäristön tapauksessa tämä tarkoittaa, että kursorin vieminen tunnuksen päälle esittää sen dokumentaation kuvan 22 mukaisesti.



Kuva 22: Dokumentaation integraatio

SQF-ohjelmointikielen mahdollisesti kattavin dokumentaatio on Bohemian Interactive Community Wiki -sivusto. Side kykenee esittämään lähes jokaisesta SQF-komennosta ja -funktioista HTML-tyyppisen dokumentaation. Nämä dokumentaatiot ovat automaattisesti generoituja Bohemia Interactive Community Wiki -sivuista Biki-parserin toimesta.

Kuvassa 23 on esitetty sekvenssikaavio dokumentaation hakemisesta Biki-parserin rakentamasta XML-tiedostosta.



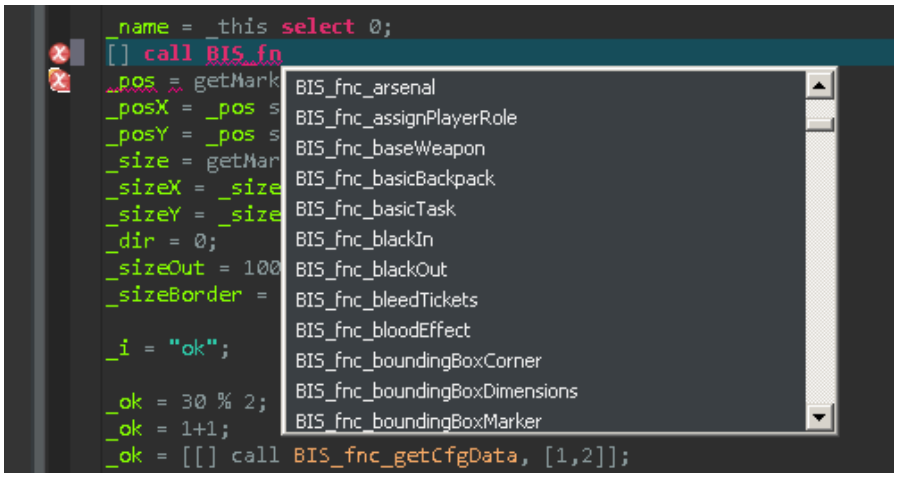
Kuva 23: Dokumentaation haku

Editori hakee dokumentaation komennolle tai funktioille kutsumalla *SqfEObjectDocumentationProvider*-luokan *getDocumentation*-funktioita. *SqfEObjectDocumentationProvider* kysyy annetulta parametrilta sen merkkijonotyyppisen nimen. *SqfEObjectDocumentationProvider* kutsuu *XmlReader*-luokan *getCallableDoc*-funktiota ja antaa parametrina nimen. *XmlReader* jäsentää XML-tiedoston ja palauttaa dokumentaation merkkijonona, joka palautetaan edelleen *SqfEObjectDocumentationProvider*-luokan *getDocumentation*-funktion paluuarvona.

11.6 Funktioiden automaattinen täydennys

SQF tarjoaa mahdollisuuden toiminnallisuuden kapselointiin funktiomäärittelyiden kautta. Tehtävät voivat määritellä itselleen globaaleja funktioita, mutta voivat myös käyttää modien globaaleja funktioita. Arma 3 -pelin modifikaatiot voivat täten käyttäytyä ohjelmistokirjaston tapaisesti. Mahdollisesti yleisin tämän tyyppinen modi on Community Based Add-ons (lyh. CBA), joka tarjoaa makroihin ja funktioihin perustuvan kirjastotuen [27].

Määriteltyjen funktioiden määrää on näennäisesti rajoittamaton. Tämä tarkoittaa sitä, että yksi modi voi määrittää satoja funktioita. Näiden funktioiden nimien tarkka muistaminen aiheuttaa haasteita SQF-ohjelmoijalle. Side kykenee helpottamaan ohjelmoijan työtä tarjoamalla automaattisen täydennyksen funktioiden määrittelyille. Kuvassa 24 on esitetty esimerkki funktion täydennyksestä Side-kehitysympäristössä.



Kuva 24: Automaattisen täytön graafinen käyttöliittymä

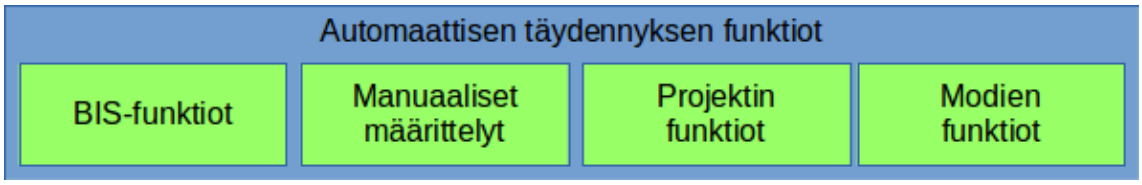
Kuten kuvasta 24 voidaan havaita, SQF-funktion tägimerkintä on merkittävällä tavalla hyödyllinen funktiokutsua kirjoittaessa.

SQF-funktioiden määrittely voi olla melko mielivaltainen, johtuen lähdekoodimuuttujan ja funktion identtisyydestä. Funktion määrittelylle on olemassa neljä tapaa:

1. Julkisen koodimuuttujan määrittely
2. Globaalin koodimuuttujan määrittely

3. Lokaalin koodimuuttujan määrittely
4. Asetustiedostossa määrittely

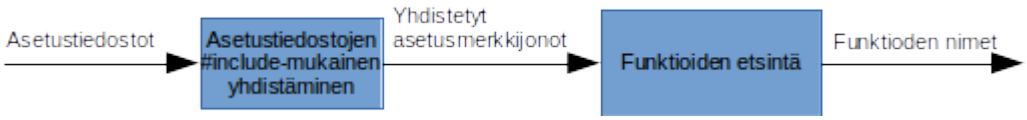
Ensimmäinen tapa mahdollistaa lähdekoodimuuttujan määrittelyn pelipalvelimella ja tämän jälkeen sen lähettämisen *publicVariable*-komennolla verkon ylitse siihen yhdistyneelle pelille (asiakasohjelmalle). Palvelimen ja pelin tehtäväkohtainen lähdekoodi ja modikokoelma voivat erota suuresti. Myös toinen ja kolmas tapa mahdollistaa mielivaltaisen tavan funktioiden määrittelylle esimerkiksi makroja tai merkkijonon ketjutusta (engl. string concatenation) hyödyntäen. Edellä mainituista syistä johtuen funktion nimeä ei voida hyödyntää syntaksin tarkistuksessa ja ainoastaan neljännellä tavalla määritellyt funktiot ovat ohjelmallisesti tuettuna automaattisen täydennyksen toimesta. Ensimmäisen ja toisen määrittelytavan funktiot voidaan kuitenkin lisätä manuaalisesti automaattiseen täydennykseen muokkaamalla projektikohtaista XML-tiedostoa. XML-formaatti mahdollistaa, että SQF-kehittäjä voi itse kirjoittaa oman funktiojäsentelijän hänen omalle funktiomäärittelytavalle. Kuvassa 25 on esitetty kaavio automaattisen täydennyksen funktiotuen laajuudesta.



Kuva 25: funktiotuen laajuus

BIS-funktioilla tarkoitetaan Arma 3 -pelin sisäisiä SQF-funktioita. Automaattisen täydennyksen funktiotukeen kuuluu BIS-funktiot, manuaalisesti määritellyt funktiot, modien asetustiedostojen funktiot ja projektin asetustiedostojen funktiot.

SQF-funktioiden havaitsemiseksi käytetään *ConfigParser*-ohjelmistokomponenttia. Kyseessä on diplomityön tekijän aikaisempi projekti. *ConfigParser* on Java-kirjasto, joka jäsentelee Arma 3 -asetustiedostojen syntaksin ja täten mahdollistaa sen ohjelmallisen muokkauksen. Kuvassa 26 on esitetty vuokaavio funktioiden etsinnän algoritmista.

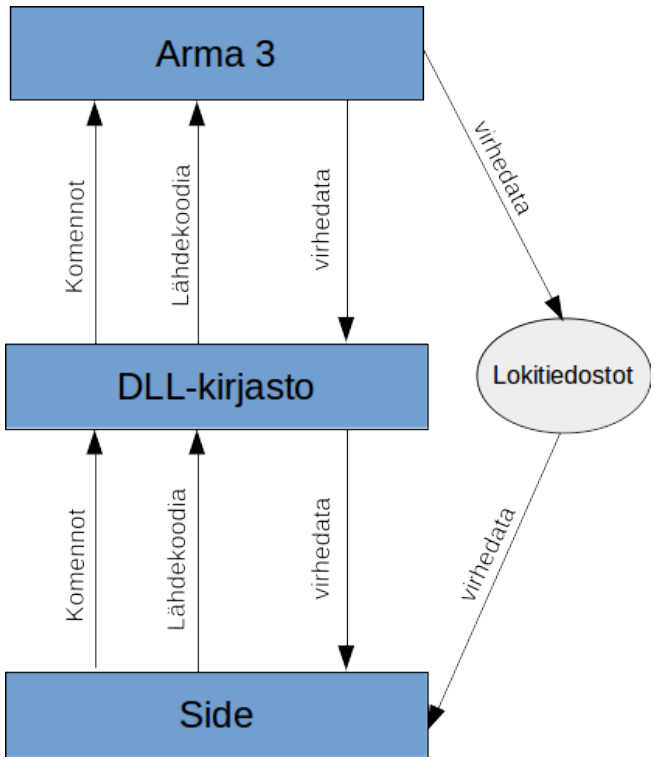


Kuva 26: Funktioiden määrittely

Algoritmi etsii modissa tai tehtävässä olevat asetustiedostot niiden päätteiden perusteella. Löydetyistä tiedostoista korvataan *#include*-käskyt niiden viittaamien tiedostojen sisällöillä. Tiedostoja, joita ei ole viitattu #include-käskyllä kutsutaan äititiedostoiksi. Äititiedostoista etsitään funktioiden nimet, jotka tallennettaan XML-tiedostoon. XML-tiedosto luetaan automaattisen täydennyksen kutsun aikana. Tehokkuussyistä johtuen XML-tiedoston sisältö luetaan tietorakenteeseen, joka päivitetään vain, jos XML-tiedoston koko muuttuu.

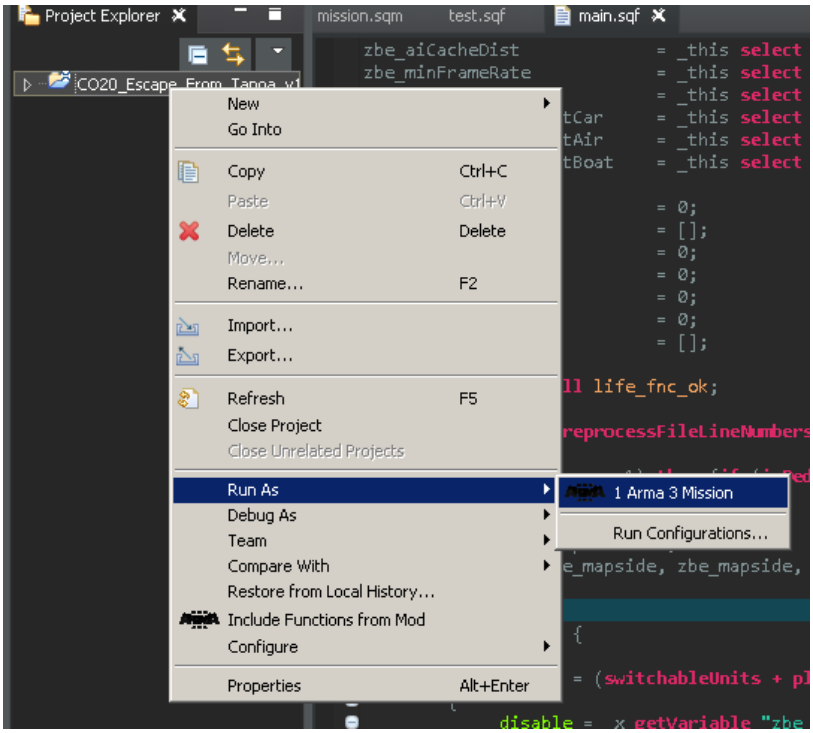
11.7 Integraatio pelin kanssa

Side kommunikoi Arma 3 -pelin kanssa lokitiedostojen ja DLL-kirjaston välityksellä. Kuvassa 27 on esitetty yleiskuvaus Side-kehitysympäristön kommunikaatiosta Arma 3 -pelin kanssa.



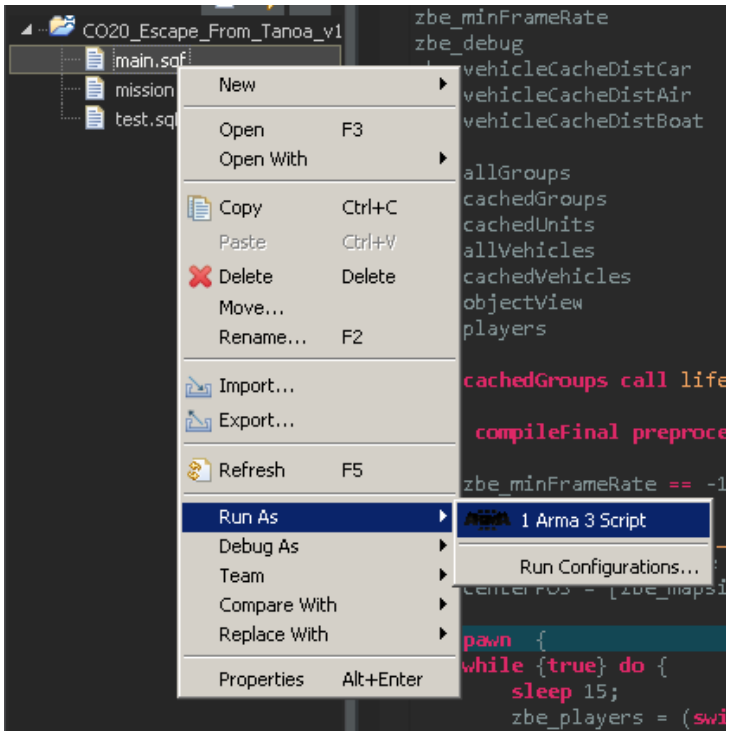
Kuva 27: Siden kommunikaatio pelin kanssa

Side kommunikoi Arma 3 -pelin kanssa siirtääkseen lähdekoodia pelin sisälle ja saadakseen virhedataa ohjelmointivirheiden osoittamiseksi. Dynaaminen kirjasto (engl. dynamic-link library, lyh. DLL) ladataan pelin käynnistyessä ja peli kommunikoi Side-kehitysympäristön kanssa sen välityksellä. DLL-kirjaston aktivoitumiseksi peli tulee käynnistää Side-kehitysympäristöstä käsin. Side-kehitysympäristössä avattu tehtävä voidaan käynnistää Arma 3 -pelissä kontekstivalikon kautta kuvan 28 osoittamalla tavalla.



Kuva 28: Arma 3 -tehtävän käynnistys

Peli lataa käyttöönsä Side-kehitysympäristön dynaamisen kirjaston, joka mahdollistaa kommunikaation pelin ja kehitysympäristön välillä. Lähdekooditiedosto voidaan kääntää ja ajaa pelissä kuvan 29 mukaisen kontekstivalikon kautta.

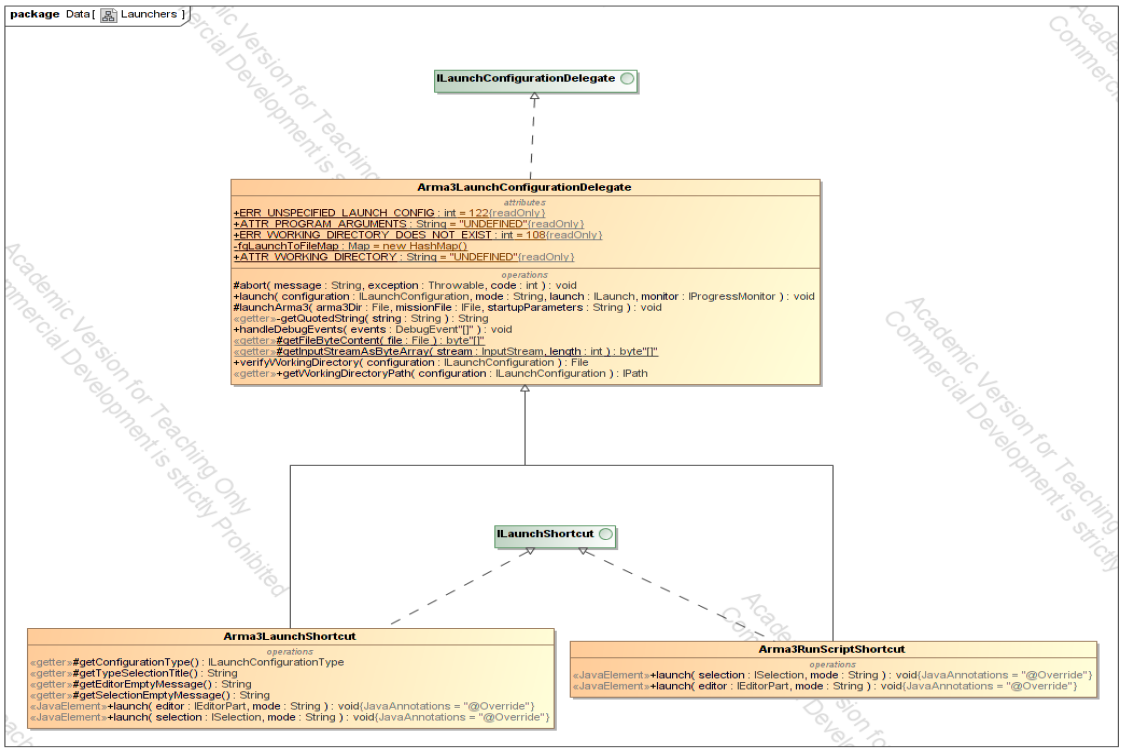


Kuva 29: Lähdekooditiedoston ajaminen

Lähetetyn lähdekooditiedoston sisältö kootaan merkkijonoksi, käännetään ja ajetaan pelin sisällä. Syntyneitä virheitä voidaan tarkastella lokitiedostoista. Side tukee integroitua lokitiedostojen tarkastelua (luku 11.7.3).

11.7.1 Käynnistysvalikoiden tekninen kuvaus

Tehtävän käynnistämiseen ja lähdekooditiedoston ajamiseen tarkoitetut valikot ovat molemmat toteutettu käyttämällä *org.eclipse.debug.ui.launchShortcuts* -laajennuspistettä. Kuvassa 30 on esitetty luokkakaavio laajennuspistettä käyttävistä luokista.

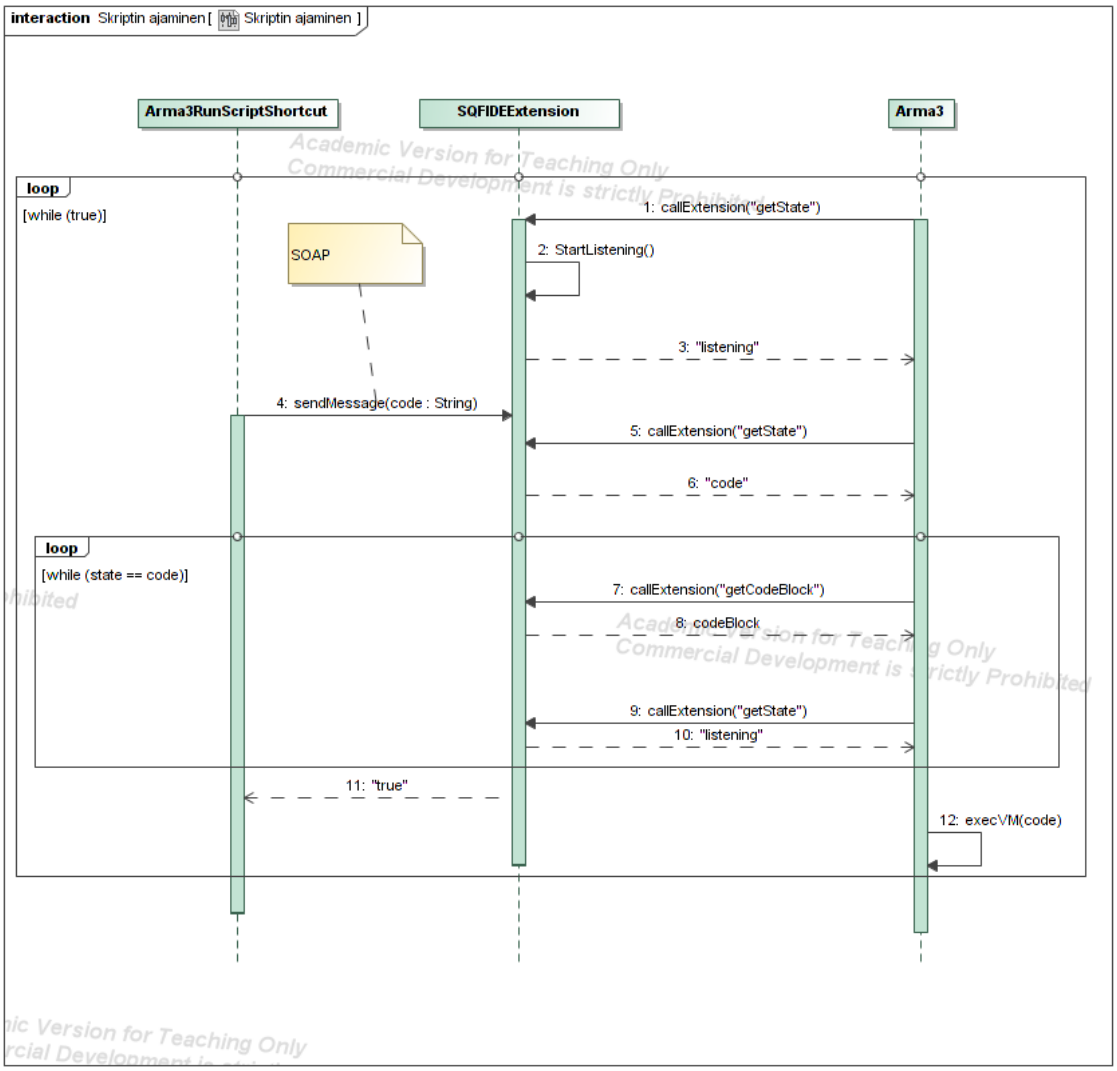


Kuva 30: Suorituksen ohjelmallinen toteutus

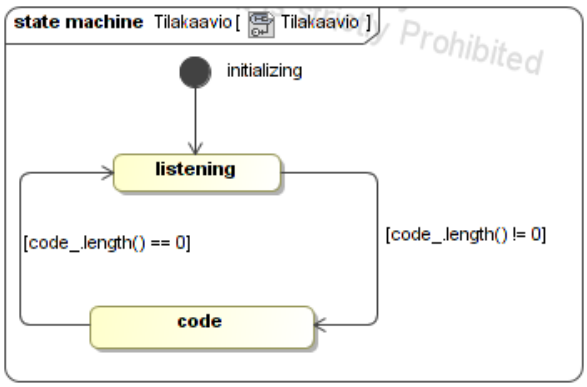
Laajennuspisteiden toteuttavat luokat ovat *Arma3LaunchShortcut* tehtävän käynnistystä varten ja *Arma3RunScriptShortcut* lähdekooditiedoston ajamista varten. Molemmat luokat perivät kantaluokan *Arma3LaunchConfigurationDelegate*, johon on toteutettu yhteiset toiminnot. Molemmat periytetyt luokat toteuttavat Eclipse-alustan *ILaunchShortcut* -rajapinnan ja *Arma3LaunchConfigurationDelegate* toteuttaa Eclipse-alustan *Ilaunch-ConfiugrationDelegate*-rajapinnan.

11.7.2 Dynaamisen kirjaston tekninen kuvaus

Kommunikaation onnistumiseksi pelin ja Side-kehitysympäristön välissä tulee olla DLL-kirjasto, jonka peli lataa käyttöönsä. Kehitetty dynaaminen kirjasto hyödyntää web service -teknologiaa kommunikaatiossa Side-kehitysympäristön kanssa. Kirjaston latushetkellä se käynnistää gSoapin avulla toteutetun *Web Service* -palvelun, joka tarjoaa *runScript*-funktion Side-kehitysympäristölle. Kyseinen funktio ottaa sisäänsä merkkijonotyyppisen muuttujan. Dynaaminen kirjasto ei kykene kutsumaan peliä itse, joten pelin sisäisesti toimiva SQF-toteutus kiertokyselee lähdekoodia siltä tietyin väliajoin. Kuvassa 31 on esitetty sekvenssikaavio lähdekoodin siirtämisestä Side-kehitysympäristöstä peliin ja kuvassa 32 on esitetty tilakaavio kirjaston toiminnasta.



Kuva 31: Komentosarjan siirtäminen Side-kehitysympäristöstä peliin.



Kuva 32: Kirjaston tilakaavio

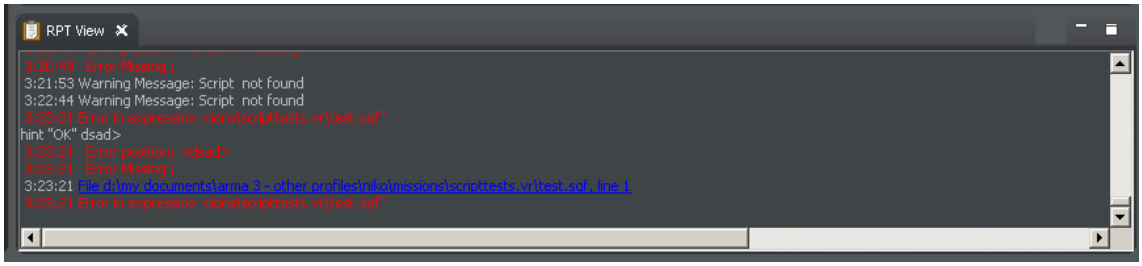
Ensimmäiseksi peli lataa dynaamisen kirjaston ja kutsuu sen *getState*-funktiota. Kirjasto palauttaa tilakseen *listening*. Side-kehitysympäristön *Arma3RunScriptShortcut*-olio lä-

hettää lähdekoodia kirjastolle. Lähdekoodin saavuttua kirjaston tilaksi asettuu *code,* joka tarkoittaa, että lähdekoodia on vastaanotettu ja se on luettavana. Pelin sisällä toimiva SQF-toteutus alkaa lukemaan lähdekoodia *getCodeBlock*-funktion avulla, mikä kerrallaan lukee kirjastolta Arma 3 pituusrajoituksien mukaisen pätkän lähdekoodia. Kun koko lähdekoodi on luettu, asettuu kirjasto uudelleen *listening*-tilaan. Luetuista lähdekoodisegmenteistä yhdistämällä luotu komentosarja ajetaan Arma 3 -pelissä *execVM*-komennolla. Kirjasto ilmoittaa Side-kehitysympäristölle lähdekoodinsiirron onnistumisesta *true*-paluuarvolla.

11.7.3 Lokitiedostojen integraatio

Arma 3 generoi lokitiedostoja, joihin kirjataan muun muassa SQF-virhe- ja varoitusilmoitukset. Näitä lokitiedostoja kutsutaan raporttitiedostoiksi tai lyhennettynä RPT-tiedostoiksi. Arma 3 voi tallentaa useamman RPT-tiedoston tallennusajan perusteella RPT-lokikansioon. RPT-tiedoston nimi sisältää sen luontiajan.

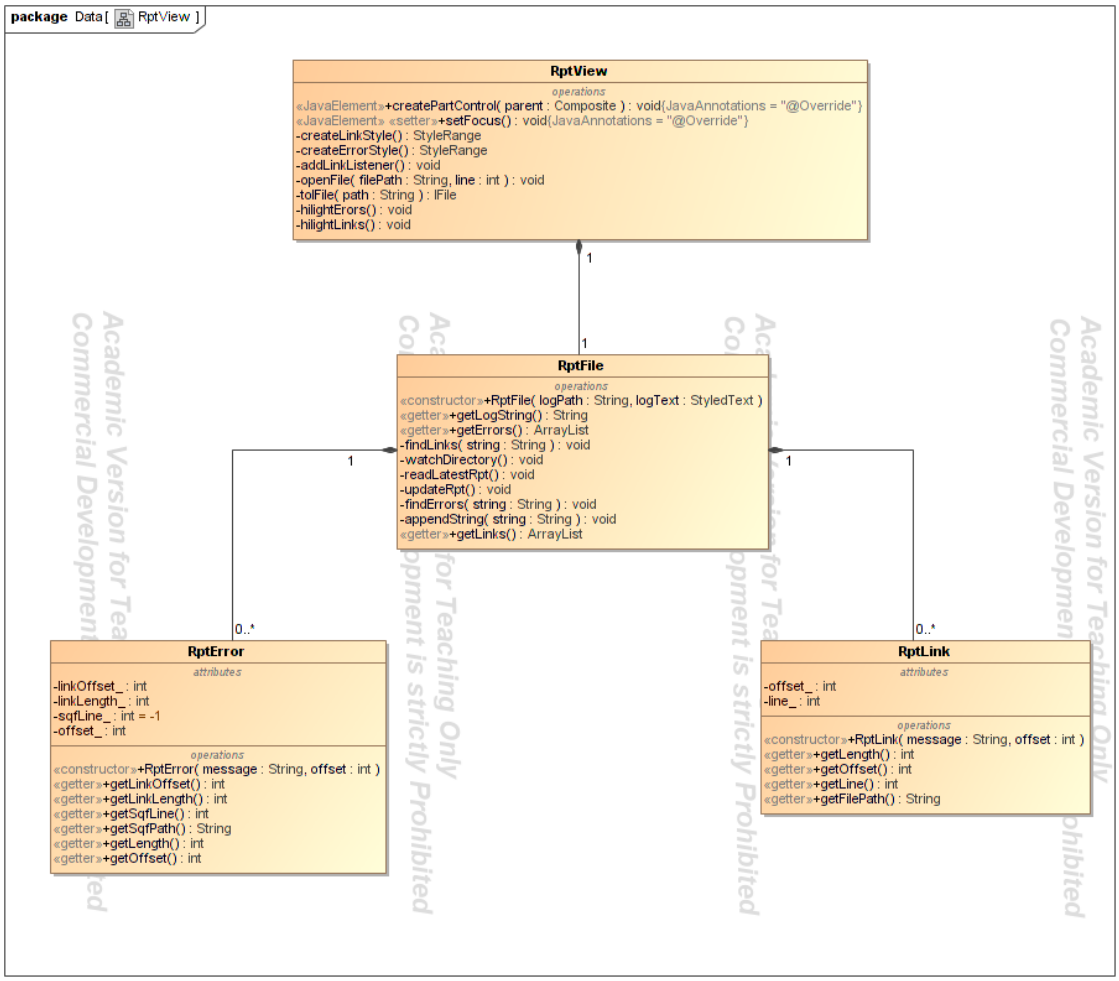
SQF-ohjelmointityön tehostamiseksi Side-kehitysympäristössä on toteutettuna loki-integraatio, jota kutsutaan RPT-näkymäksi. Kuvassa 33 on esitetty RPT-näkymä.



Kuva 33: RPT-näkymä

RPT-näkymän pääasiallisena tarkoituksena on saattaa ohjelmointivirheistä tulleet varoitus- tai virheilmoitukset SQF-ohjelmoijan tietoon mahdollisimman nopeasti ja selkeästi osoittaa ohjelmoija virheen sijaintiin. RPT-näkymä värittää virheilmoitukset punaisella ja tiedostoihin tehdyt viitteet, kuten tiedoston nimi ja rivinumero, muunnetaan linkkimuotoon. Linkkiä painamalla linkattu SQF-lähdekooditiedosto avataan Side-kehitysympäristössä ja kursori ohjataan RPT-tiedostossa ilmoitetulle riville.

RPT-näkymä on toteutettu Side-kehitysympäristön käyttöliittymäominaisuudessa. Kuvassa 34 on esitetty luokkakaavio RPT-näkymän toteutuksesta.



Kuva 34: RPT-näkymän luokkakaavio

RptFile-luokan tarkoituksena on käsitellä RPT-tiedostoja. *RptFile* ottaa rakentajassaan parametrina polun RPT-lokikansioon ja avaa sieltä uusimman RPT-tiedoston käsittelyä varten. *RptFile*-olio jäsentää virherivit ja tiedostolinkit. Virheriveistä muodostetaan *RptError*-olioita, joihin tallennetaan tarvittavat tiedot virheen väritystä varten. Tiedostoviitteistä muodostetaan *RptLink*-olioita, joihin tallennetaan tarvittavat tiedot viitteiden väritystä varten. *RptLink* jäsentää tiedostoviitteestä tiedoston rivin ja polun. *RptView* piirtää RPT-tiedoston sisällön *RptFile*-olion avulla. Se kertoo ohjelman käynnistysvaiheessa *RptFile*-olioille RPT-lokikansion sisällön. *RptView* näyttää *RptFile*-olion sisällön, värittää *RptError*-oliota vastaavat rivit ja asettaa linkit *RptLink*-olioita vastaaville merkkijonoille.

11.8 Käyttöliittymä

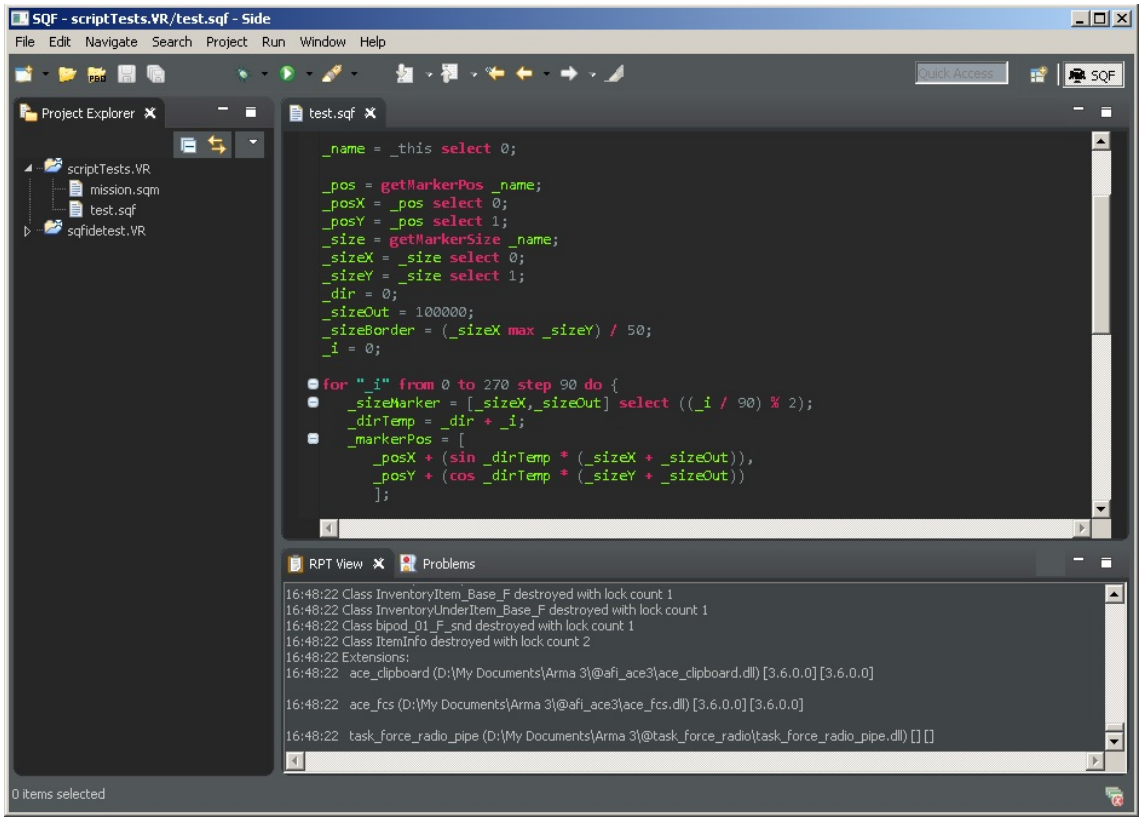
Side-kehitysympäristön käyttöliittymä noudattaa Eclipse-alustalle toteutettujen integroitujen kehitysympäristöjen tyypillistä ulkoasua. SQF-ohjelmoinnin yksinkertaisuuden ta-

kia, joitakin toimintoja on pyritty tuomaan paremmin näkyväksi käyttäjälle. Näitä ovat muun muassa projektin avaaminen, joka tavanomaisesti tapahtuisi tuo-toiminnon (engl. import) kautta, mutta Side-kehitysympäristössä se voidaan tehdä myös avaa-kuvakkeen välityksellä työkalupalkista.

Tämä luku koostuu kolmesta alaluvusta. Alaluvussa 11.8.1 kerrotaan Side-kehitysympäristön päänäkymästä; alaluvussa 11.8.2 esitellään asetusvalikot; alaluvussa 11.8.3 kerrotaan, miten Arma 3 -tehtävä avataan Side-kehitysympäristössä. Jokaisessa alaluvussa keskitytään ensimmäiseksi käyttökuvaukseen ja sen jälkeen toteutuksen tekniseen kuvaukseen. Tästä luvusta on jätetty pois ne käyttöliittymän osuudet, jotka liittyvät integraatioon pelin kanssa. Nämä osuudet on kuvattu luvussa 11.7. PBO-arkistojen avaamiseen liittyvä käyttöliittymä on kuvattu luvussa 11.4.

11.8.1 Päänäkymä

Kuvassa 35 on esitetty Side-kehitysympäristön päänäkymä. Kyseessä on oletusnäkymä, joka näkyy käyttäjälle ilman ohjelman asetuksien muuttamista. Demonstraatiosyistä näkymässä on avattuna kaksi Arma 3 -tehtävää.



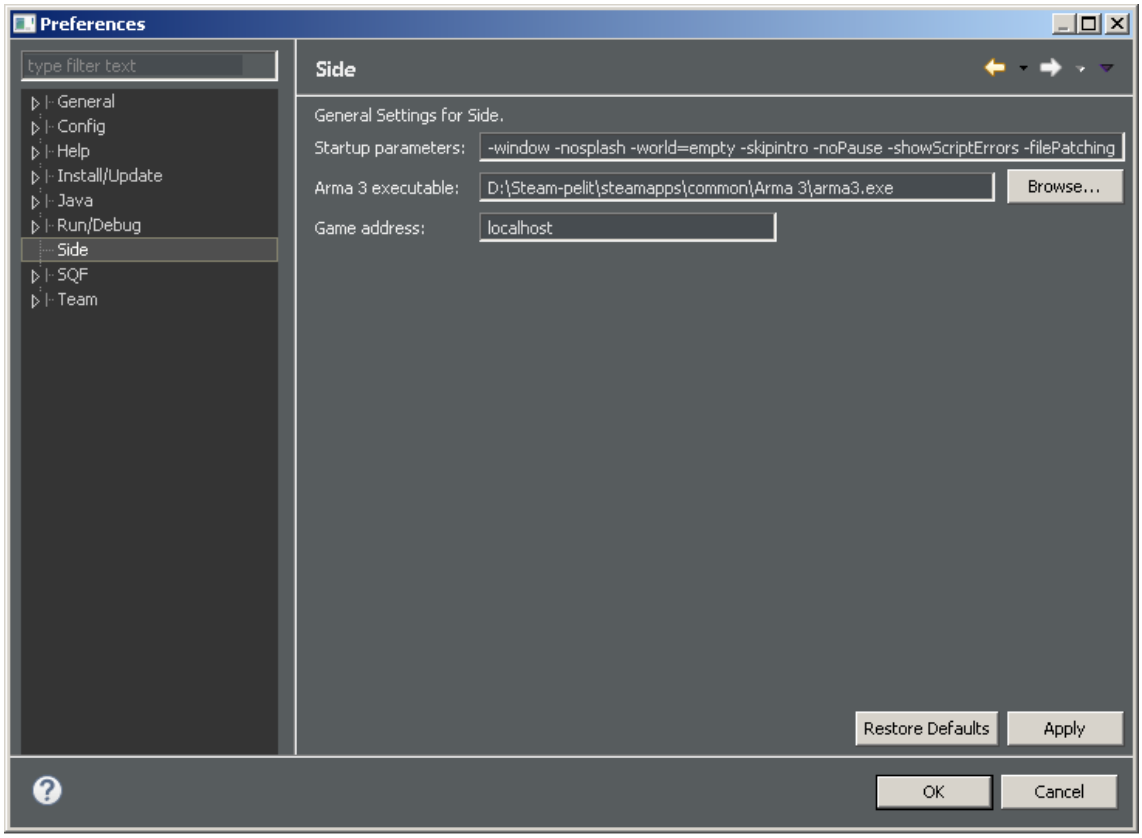
Kuva 35: Päänäkymä

Ohjelman yläreunassa, työkalupalkissa, on keltaisina kansioina pikakuvakkeet Arma 3 -tehtävän avaamiseen. Avaa-kuvake, jonka alakulmassa on teksti ”pbo”. Tämä mahdollistaa tehtävän avaamisen PBO-arkistosta ja ilman tekstiä oleva avaa kansiomuotoisen

tehtävän. Vasemmassa reunassa oleva Project Explorer näyttää auki olevat projektit. Nykyisessä Side-versiossa tämä tarkoittaa avoinna olevia tehtäviä, mutta se voi tulevaisuudessa tarkoittaa myös kehitettäviä modeja. Lähdekoodinäkymä on päänäkymässä oikealla keskellä ja vie suurimman osan näyttötilasta. Alhaalla oikealla näkyy RPT-näkymä. Vaihtoehtoisesti RPT-näkymän paikalla voidaan näyttää ongelmanäkymä (engl. problems view), josta voidaan tarkastella ohjelmointivirheitä. Päänäkymän oikeassa yläkulmassa näkyy SQF-perspektiivi valittuna.

11.8.2 Asetusnäkymä

Kuvassa 36 on esitetty Side-kehitysympäristön asetusnäkymä. Asetukset sijaitsevat Eclipse-alustalle tyypillisessä valikossa, jonka saa auki painamalla kehitysympäristön ylävalikosta window → Preferences.



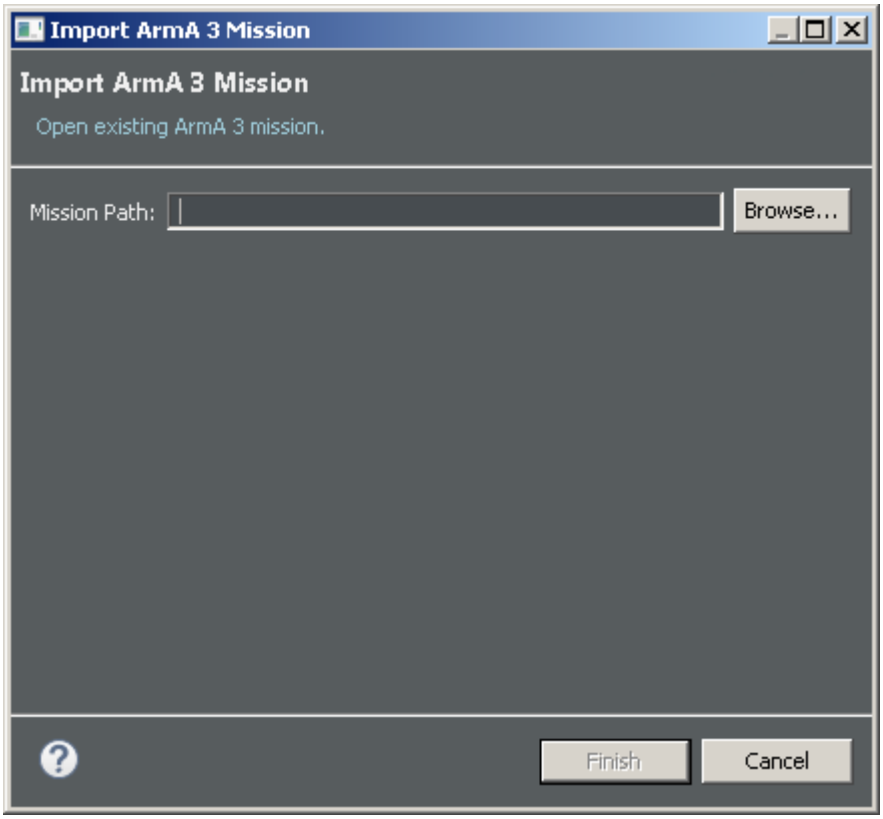
Kuva 36: Side-kehitysympäristön asetukset

Asetuksista voi muutta käynnistysparametreja, Arma 3 -pelin käynnistystiedoston sijaintia ja pelin verkko-osoitetta. Käynnistysparametreja käytetään, kun peli käynnistetään Side-kehitysympäristön kautta. Oletuksena niihin on sisällytetty parametreja, joita yleensä käytetään pelikehityksen aikana. Pelin osoite määrittää osoitteen mihin otetaan yhteys, kun Side kommunikoi pelin kanssa Web service -teknologian avulla.

Asetusnäkymä on toteutettu laajentamalla *org.eclipse.ui.preferencePages*-laajennuspistettä, jonka toteuttaa *PreferencePage*-luokka. Lisäksi asetuksien oletusarvot asetetaan *PreferenceInitializer*-luokassa, joka laajentaa laajennuspistettä *org.eclipse.core.runtime.preferences*. *PreferenceInitializer* pyrkii etsimään oletusarvot pelin käynnistystiedostolle Windows-käyttöjärjestelmän rekisteristä. Muut oletusarvot ovat ennalta määriteltyjä vakiota.

11.8.3 Arma 3 -tehtävien avausvalikko

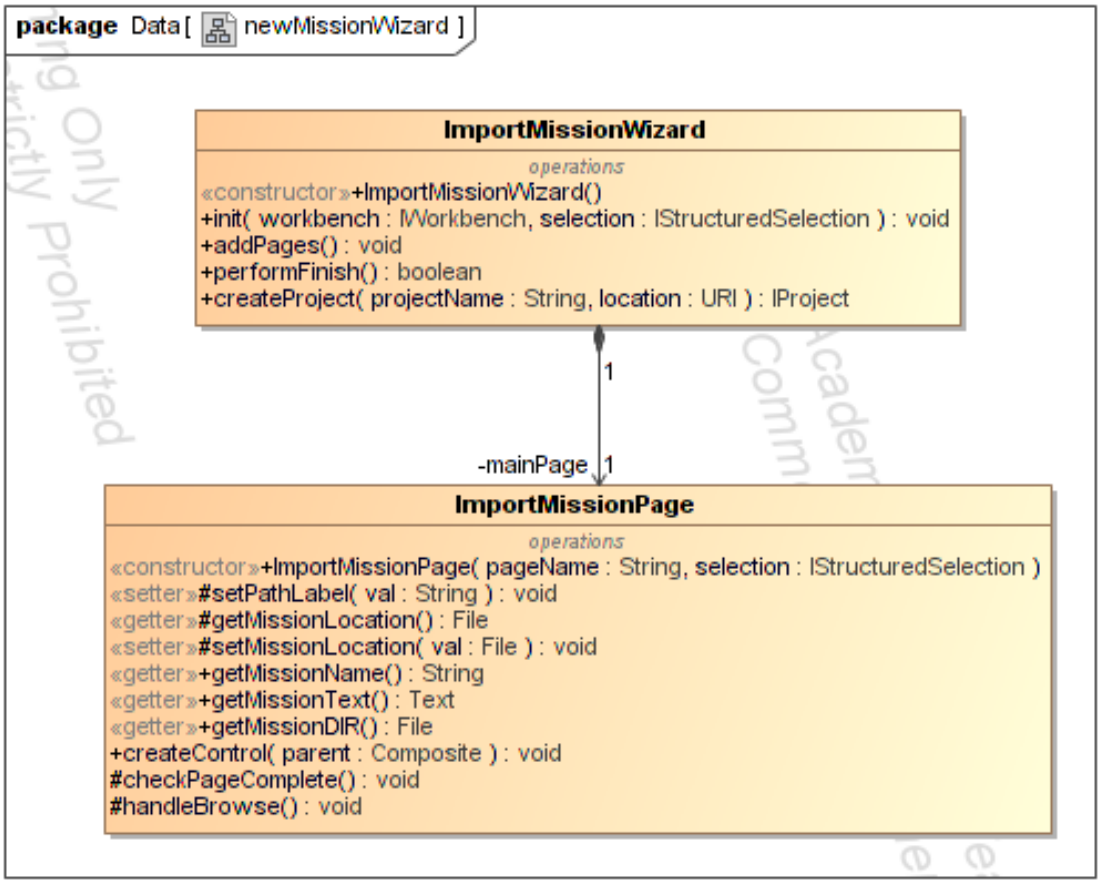
Kuvassa 37 on esitetty avustaja, jonka avulla Arma 3 -tehtävä avataan. Toiminto on Eclipse-ympäristön tuontiominaisuus (engl. import). Kyseisen ikkunan saa auki joko tyypillisen Eclipse *import*-valikon kautta tai painamalla avaa-kuvaketta Side-kehitysympäristön työkalupalkista.



Kuva 37: Import Mission Wizard

Painamalla *Browse* käyttäjä voi valita Arma 3 -tehtäväkansion. Side tarkistaa sisältyykö kansioon vaaditut tiedostot. Jos vaaditut tiedostot ovat kansiossa, *Finish*-näppäin aktivoituu. *Finish*-näppäintä painamalla tehtävä avautuu Side-kehitysympäristön työtilassa.

Arma 3 -tehtävän avaaminen on toteutettu laajentamalla *org.eclipse.ui.importWizards*-laajennuspistettä. Laajennuspisteen toteuttaa *ImportMissionWizard*-luokka. Kuvassa 38 on esitetty luokkakaavio toimintoon liittyvistä luokista.



Kuva 38: Arma 3 -tehtävän avaaminen, luokkakaavio

Toiminto koostuu kahdesta luokasta: *ImportMissionWizard* ja *ImportMissionPage*. *ImportMissionWizard* sisältää yhden sivun, jonka toteuttaa *ImportMissionPage*. *ImportMissionWizard* toteuttaa tehtävän avaamiseen liittyvät toimenpiteet. *ImportMissionWizard* piirtää avustajan ainoan sivun ja tarkistaa, että tehtäväkansio sisältää tarvittavat tiedostot, minkä perusteella se aktivoi *Finish*-näppäimen.

11.9 Päätelmät

Side-kehitysympäristön toteutuksessa pyrittiin maksimoimaan korkean prioriteetin ominaisuudet samalla säilyttäen ohjelma käytettävänä kokonaisuutena. Ensisijaisena tavoitteena oli vastata kolmanteen tutkimuskysymykseen.

Side koostuu kolmesta ohjelmistokomponentista: käyttöliittymästä, Config-editorista ja SQF-editorista. Käyttöliittymän toteutuksessa hyödynnettiin puhdasta Eclipse-kehystä ja editorit kehitettiin Xtext-kehystä hyödyntäen. Eclipse-alustaa laajennetaan sen laajennuspisteiden kautta. Osa käytetyistä laajennuspisteitä löytyivät ArmaDev-lisäosan laajennusmäärittelyistä ja osa Eclipse-alustan dokumentaatiosta. Eclipse-alustan laajentaminen osoittautui helpoksi, mutta vain osalle luokista löytyi abstrakti kantaluok-

ka, joka toteuttaa geneerisimmät ominaisuudet. Esimerkkinä geneerisestä kantaluokasta oli *FieldEditorPreferencePage,* joka yksinkertaisti merkittävästi asetuksien toteutusta.

Xtext-kehys mahdollisti lähdekoodimäärältään pienen editorin toteutuksen. SQF-ohjelmointikielen Xtext-kielioppimäärittely koostuu noin 3000 merkistä, jos siinä listattuja komentoja ei oteta huomioon ja Config-kielioppimäärittely on tätäkin lyhyempi. Kielioppimäärittelyiden toteuttaminen osoittautui kuitenkin aikaa vieväksi ja Xtext-kehyksen antamat virheilmoitukset vaikeaselkoisiksi. Käytännössä virheilmoitukset osoittavat ANTRL-kielioppiin, jonka yhteyttä Xtext-kielioppiin ei voida aina helposti selvittää. Xtext-kielioppi tulee olla myös semanttisen määrittelyn kanssa yhteensopiva, minkä takia joitakin syntaksimielessä turhia määrittelyitä joudutaan tekemään. Xtext-kielioppimäärittelyiden voidaan kuitenkin olettaa olevan ylläpidettävämpiä kuin ohjelmallisesti toteutetut kielentunnistajat.

Xtext-kehyksen hyödyntämisessä ilmeni kaksi pääongelmaa. Ensinnäkin leksikaalisen analysaattorin Java-luokat kasvoivat SQF-komentojen suuren määrän takia liian suuriksi, minkä takia Java-tulkki kieltäytyi käynnistämästä kielentunnistajaa. Ongelma ratkaistiin siirtämällä SQF-komentomäärittelyt validoijaan. Ensimmäinen pääongelma kuitenkin johti toiseen pääongelmaa, joka oli seuraus SQF-komentojen syntaktisesta identtisyydestä globaaleiden muuttujaviitteiden kanssa. Kielentunnistajan linkkeri ja validoija toimivat toisistaan itsenäisesti, minkä takia SQF-komentoa vastaava avainsana aiheutti viitevirheen editorissa. Ongelma ratkaistiin ylikirjoittamalla Xtext-kehyksen linkkeri.

Xtext-kieliopin merkittävä ongelma on C-tyyppisten include- ja makromäärittelyjen tuen puute. Include-ongelma ratkaistiin hyödyntämällä globaaleita lohkoja. Tällöin editori ei ilmoita linkkaukseen liittyviä valevirheitä, mutta voi olla ilmoittamatta joistakin linkkausvirheistä. Vaatimusmäärittelyn mukaan tämä on sallittavaa. Makromäärittelyiden kohdalla ongelmaa ei diplomityön aikana ratkaistu.

Kielioppimäärittelyiden suunnittelussa hyödynnettiin testivetoista kehitystä. Käytännössä ensimmäiseksi kehitettiin raakamäärittely, jota sovellettiin yhteisön tekemiin SQF-lähdekooditiedostoihin. Tavoitteena oli valevirheiden poistaminen. Löytyneestä valevirheestä kirjoitettiin yksikkötesti, joka korjattiin jälkeenpäin. Xtext-kehyksen yksikkötestausominaisuudet ovat puutteellisia, minkä takia hyödynnettiin Xtext Utils -kirjastoa, joka tarjoaa laajemman yksikkötestaustoiminnallisuuden.

Luvussa 3.4 esitettiin kymmenen ominaisuutta, jotka löytyvät suosituista integroiduista kehitysympäristöistä. Taulukossa 9 on esitetty Side-kehitysympäristön ominaisuudet suhteessa kilpaileviin SQF-editoreihin.

Taulukko 9: Side suhteessa kilpailijoihinsa

	Ohjelma			
Ominaisuus	Notepad++	Squint	ArmaDev	Side
1. Refaktorointi	X		X	X
2. Automaattinen täydennys	X	X	X	X
3. Sisällön avustaja		X	X	X
4. Syntaksikorotus	X	X	X	X
5. Syntaksin tarkistus		X		X
6. Korjauksen ehdotus		X		X
7. Virheidenetsintätuki			X	X
8. Käyttöliittymäeditori				
9. Laajennettavuus				X
10. Ohjelman ajaminen				X

Side toteuttaa kaikki ominaisuudet paitsi käyttöliittymäeditorin. Korjauksen ehdotus toteutuu Xtext-kehyksen toiminnallisuuden kautta. RPT-lokituki mielletään osaksi virheidenetsintää niin kuin myös ArmaDev-ohjelmiston tapauksessa. Side-kehitysympäristön lähdekoodin avoimuus täyttää laajennettavuuden kriteerit.

Side ei ole saavuttanut diplomityön aikana julkaisukelpoisuutta. Tärkeimpänä syynä tähän oli syntaksin tarkastajan ilmoittamat valevirheet. Osa valevirheistä oli helposti korjattavissa, mutta aikarajoituksien takia niiden korjaus jätettiin jatkokehitykseen. Esimerkiksi SQF mahdollistaa muuttujan määrittelyn tavalla, jota kielioppimäärittely ei ollut mallintanut. Kehityksen aikana havaittiin, että SQF ei sovellu hyvin tarkalle syntaksin tarkistukselle, minkä takia tietyt valevirheet sallitaan. Tavoitteena on kuitenkin kehittää helppo tapa piilottaa valevirheet. Myös Side-kehitysympäristön käyttöliittymä on ulkonäöllisesti viimeistelemätön samoin kuin lähdekoodikin. Side-kehitysympäristön toteutus vastaa kuitenkin kolmanteen tutkimuskysymykseen.

12 JOHTOPÄÄTÖKSET

Tässä diplomityössä käsiteltiin integroitujan kehitysympäristöjä. Tavoitteena oli tutkia olemassa olevia integroituja kehitysympäristöjä käytön, rakenteen ja historian suhteen, ja kehittää saadun tiedon perusteella uusi integroitu kehitysympäristö SQF-ohjelmointikielelle.

Ensimmäinen tutkimuskysymys oli: ”Miten ohjelmistokehittäjät hyödyntävät integroituja kehitysympäristöjä?”. Ensimmäiseen tutkimuskysymykseen vastattiin kirjallisuuden avulla luvussa 2. Luvussa havaittiin, että refaktorointikomentojen käyttö on lähes yhtä suosittua kuin tavanomaisten tekstinkäsittelykomentojen käyttö. Myös automaattinen täydennyksen havaittiin olevan erittäin suosittu ominaisuus. Integroituja kehitysympäristöjä käytettiin kaiken kokoisissa organisaatioissa pääasiassa ohjelmistokehittäjien toimesta. Integroitujen kehitysympäristöjen käyttö on kuitenkin yleistä myös muissa ammateissa. Kokonaisuudessaan voidaan tehdä johtopäätös, että integroitujen kehitysympäristöjen ominaisuudet koetaan erittäin hyödylliseksi ohjelmistokehityksessä.

Toinen tutkimuskysymys oli: ”Mitä käyttäjävaatimuksia tulisi olla SQF- ja Arma 3 -kohtaisella integroidulla kehitysympäristöllä?”. Tähän tutkimuskysymykseen vastattiin luvuissa 2 ja 3. Vastausta varmennettiin toteutusosan avulla luvuissa 9, 10 ja 11. Näissä luvuissa havaittiin, että lähdekoodin refaktorointikohtaisten ominaisuuksien kehittäminen on lähes yhtä tärkeää kuin perinteisten tekstinkäsittelyominaisuuksien. Luvussa 3 havaittiin, että SQF-ohjelmoijat eivät juurikaan toivoneet perinteisten integroitujen kehitysympäristöjen ominaisuuksista poikkeavia ominaisuuksia. Side-kehitysympäristöön kuitenkin toteutettiin integraatio Arma 3 -pelin kanssa. Käytännössä tämä tarkoitti lähdekoodiprojektien käynnistämistä pelin kanssa ja lähdekoodin ajamista pelin sisällä integroidusta kehitysympäristöstä käsin. Näiden vaatimuksien täyttämiseen päädyttiin, koska niiden voidaan ajatella helpottavan ohjelmistokehitystyötä ja ne mielletään osaksi ohjelman ajamista, joka on yksi yleisistä integroidun kehitysympäristöjen ominaisuuksista. Väitteen validius voitaisiin myöhemmin todeta käyttäjätutkimuksella.

Diplomityön aikana Side ei saavuttanut julkaisukelpoisuutta. Ratkaisuun päädyttiin, koska testivetoisen kehityksen aikana löytyi edelleen SQF-tiedostoja, joissa ilmeni valevirheitä. Valevirheiden korjaamisen esteenä olivat lähinnä aikarajoitukset. Side-kehitysympäristön lähdekoodia ei myöskään ehditty viimeistelemään aikarajoituksien si-

sällä. SQF-editoreiden käyttäjätutkimuksen aikana havaittiin, että virheellisen ohjelman julkaisu ei ole kannattavaa (vrt. Squint).

Kolmas tutkimuskysymys oli: ”Miten Eclipse-alustan päälle voidaan kehittää integroitu kehitysympäristö?”. Luvuissa 9, 10 ja 11 vastattiin tähän kysymykseen. Sidekehitysympäristön suunnittelussa hyödynnettiin epätavanoinaista laajennussuunnittelumenetelmää, jonka avulla pyrittiin selvittämään, mitä rapapintoja tulee hyödyntää, että ohjelmistovaatimukset täyttyvät. Kehitystyön aikana tämä havaittiin toimivaksi ratkaisuksi, mutta joitakin laajennuspisteitä jouduttiin lisäämään alkuperäiseen suunnitelmaan. Eclipse-alustan lisäksi hyödynnettiin sen päällä toimivaa Xtext-kehystä. Kyseessä oli työkalu ohjelmointikielien ja niiden editoreiden toteutukseen. Xtext-kehyksen havaittiin olevan hyvä työkalu yksinkertaisen ohjelmointikielen editorin toteuttamiseen. Xtext-kehyksen rajoitukset asettavat kuitenkin tiettyjä rajaehtoja ohjelmointikielen soveltuvuudelle. Näistä rajoituksista seuraavat osoittautuivat ongelmallisiksi teknisen osuuden toteutuksessa: leksikaalisen analysaattorin monimutkaisuuden rajoitukset [28], C-tyyppisen include-ominaisuuden ja makrotuen puute. Include- ja makrotuen puutteen perusteluna on esitetty, että ne eivät ole hyvän ohjelmointikielen ominaisuuksia, joten niiden tukea ei ole Xtext-kehykseen toteutettu. Väitteen taustalla on ajatus, että Xtext-kehys on pääasiassa työkalu uuden ohjelmointikielen toteuttamiseen, eikä työkalu olemassa olevan ohjelmointikielen editorin toteutukseen.

Käytön aikana havaittiin, että tiedoston avaaminen saattaa isoissa projekteissa kestää useita sekunteja, mikä tosin voi johtua myös tehottomasta kielioppimäärittelystä. Ohjelmiston laadulliset tehokkuusvaatimukset kuitenkin täyttyivät. Xtext-kehyksen laajennettavuus on Google Guice -riippuvuusinjektion takia erittäin hyvä, mikä käytännössä mahdollistaa lähes kaikkien kielioppimäärittelyongelmien kiertämisen. Lukuun ottamatta makrotuen puutetta kaikki ongelmat pystyttiin kiertämään diplomityön toteutushetkellä . Kokonaisuudessaan Eclipse ja Xtext soveltuvat hyvin integroidun kehitysympäristön toteuttamiseen, kunhan kohdekieli täyttää edellä mainitut kolme reunaehtoa: yksinkertaisuus, ei makroja ja ei C-tyyppisiä includeja. On hyvin mahdollista, että tämä yhdistelmä mahdollistaa nopeimman integroidun kehitysympäristön toteutuksen. Diplomityössä havaittiin, että Eclipse-alustaan perustuva integroitu kehitysympäristö voidaan kehittää ketteriä menetelmiä hyödyntäen.

LÄHTEET

[1] G. Murphy, M. Kersten and L. Findlater, "How are Java software developers using the Elipse IDE?", *IEEE Software,* vol. 23, no. 4, sivut: 76-83, 2006.

[2] B. Konsynski, J. Kottemann, J. Nunamaker and J. Stott, "plexsys -84: An Integrated Development Environment for Information Systems", *Journal of Management Information Systems*, vol. 1, no. 3, sivut: 64-104, 1984.

[3] P. S. Newman. ”Towards an integrated development environment”, *IBM Systems Journal*, vol. 23, no. 4, sivut: 81-107, 1982

[4] A. Patrizio, "The History of Visual Development Environments", *Mendix*, 2013. [WWW]. Saatavilla: http://www.mendix.com/think-tank/the-history-of-visual-development-environments-imagine-theres-no-ides-its-difficult-if-you-try/. [Viitattu: 6.11.2016].

[5] 'A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development', *The Universal Journal of Applied computer Science and Technology,* vol. 1, no. 1, sivut: 9-15, 2011.

[6] G. Murphy, M. Kersten and L. Findlater, 'How are Java software developers using the Elipse IDE?', *IEEE Software*, vol. 23, no. 4, sivut: 76-83, 2006.

[7] "NetBeans IDE - Overview", *Netbeans.org.* [WWW]. Saatavilla: https://netbeans.org/features/index.html. [Viitattu: 08.05.2015].

[8] J. McAffer and J. Lemieux, *Eclipse Rich Client Platform.* Upper Saddle River, NJ: Addison-Wesley, 2006.

[9] *Xtext Documentation.* 2014.

[10] R. Gronback, "Eclipse Modeling Project", *Eclipse.org*. [WWW]. Saatavilla: http://www.eclipse.org/modeling/emf/. [Viitattu: 05.05. 2015].

[11] H. Postigo, 'Of Mods and Modders: Chasing Down the Value of Fan-Based Digital Game Modifications', *Games and Culture*, vol. 2, no. 4, sivut: 300-313, 2007.

[12] C. Cobb, *Making sense of agile project management.* Hoboken, NJ: Wiley, 2011.

[13] P. Measey, *Agile Foundation: Principles, Practices and Frameworks*. Swindon: BCS, 2015.

[14] "ArmaDEV – -FHQ-", *Friedenhq.org.* [WWW]. Saatavilla: http://friedenhq.org/?page_id=21. [Viitattu: 01.11.2015].

[15] M. Kuniavsky, E. Goodman and A. Moed, *Observing the user experience*. Waltham, MA: Morgan Kaufmann, 2012.

[16] 'Squint - Mac's ArmA tools', *sites.google.com*. [WWW]. Saatavilla: https://sites.google.com/site/macsarmatools/squint. [Viitattu: 01.11.2015].

[17] J. Team, "Eclipse Java development tools (JDT) Overview", *Eclipse.org*. [WWW]. Saatavilla: http://www.eclipse.org/jdt/overview.php. [Viitattu: 02.12.2015].
[18] 'ArmADev Eclipse Plugin - ARMA 2 & OA : Community Made Utilities', *Bohemia Interactive Forums*. [WWW]. Saatavilla: https://forums.bistudio.com/topic/118498-armadev-eclipse-plugin/. [Viitattu: 02.11.2015].
[19] 'Squint - the sqf editor and error-checker - ARMA 2 & OA : Community Made Utilities', *Bohemia Interactive Forums*. [WWW]. Saatavilla: https://forums.bistudio.com/topic/101921-squint-the-sqf-editor-and-error-checker/. [Viitattu: 02.11.2015].
[20] 'ArmA 3 Notepad++ Syntax Highlighting - ARMA 3 - GENERAL', *Bohemia Interactive Forums*. [WWW]. Saatavilla: https://forums.bistudio.com/topic/138891-arma-3-notepad-syntax-highlighting/. [Viitattu: 02.11.2015].
[21] 'Issues - squint - DH: (ARMA) Development Unraveled', *dev.withsix.com*. [WWW]. Saatavilla: http://dev.withsix.com/projects/squint/issues. [Viitattu: 02.11.2015].
[22] T. Parr, *The definitive ANTLR reference*. Raleigh, N.C.: Pragmatic Bookshelf, 2007.
[23] R. Warner and R. Harris, *The definitive guide to SWT and JFace*. Berkeley, CA: Apress, 2004.
[24] F. Turbak, D. Gifford and M. Sheldon, *Design concepts in programming languages*. Cambridge, Mass.: MIT Press, 2008.
[25] "Visual Studio 2015 Platform Targeting and Compatibility", 2016, v*isualstudio.com*. [WWW]. Saatavilla: https://www.visualstudio.com/en-us/productinfo/vs2015-compatibility-vs. [Viitattu: 16.5.2016].
[26] "Open Source", *Netbeans.org*. [WWW]. Saatavilla: https://netbeans.org/about/os/. [Viitattu: 16.5.2016].
[27] "Community Base addons A3 - Miscellaneous - Armaholic", *Armaholic.com*, 2016. [WWW]. Saatavilla: http://www.armaholic.com/page.php?id=18767. [Viitattu: 08.09.2016].
[28] ”Bug 458705 - [Serializer] Sequencer init method/create gets too big for some grammars”, *Bugs.eclipse.org,* 2015 [WWW]. Saatavilla: https://bugs.eclipse.org/bugs/show_bug.cgi?id=458705. [Viitattu: 31.10.2016].
[29] "A Brief History of NetBeans", *Netbeans.org*. [WWW]. Saatavilla: https://netbeans.org/about/history.html. [Viitattu: 05.11.2016].
[30] "History", *Antlr2.org*, 2005. [WWW]. Saatavilla: http://www.antlr2.org/history.html. [Viitattu: 06.11.2016].
[31] B. Henderson-Sellers and J. Edwards, "The object-oriented systems life cycle", *Communications of the ACM,* vol. 33, no. 9, sivut: 142-159, 1990.
[32] "Notepad++ SQF Syntax Highlighting and Auto Completion - Tools - Armaholic", *Armaholic.com*, 2016. [WWW]. Saatavilla: http://www.armaholic.com/page.php?id=8680. [Viitattu: 09.11.2016]

[33] "Visual Studio IDE", *Msdn.microsoft.com*. [WWW]. Saatavilla: https://msdn.microsoft.com/en-us/library/dn762121.aspx. [Viitattu: 14.11.2016].